Les nouveaux enjeux de la réification

A la recherche de coïncidences

François Pachet

Sony-CSL
6, rue Amyot
75005 Paris
pachet@csl.sony.fr

RÉSUMÉ. *Je propose ici un bilan personnel des « années objets » et une perspective sur les nouveaux enjeux de la programmation / représentation par objets. En tentant de décrire une certaine approche de la recherche - l'approche « Perrot » - je voudrais montrer que le facteur décisif à la fois de l'essor et de l'épuisement du domaine tient essentiellement à la nature des applications et des exemples qui ont nourri – puis cessé de nourrir – la recherche. Je propose dans un deuxième temps une perspective basée sur le nouveau paysage offert par la numérisation en masse de la culture occidentale, rien de moins. Cette numérisation produit de nouvelles classes de problèmes, qui en retour exercent une pression naturelle pour modéliser un nouveau réel et inventer de nouvelles technologies. Si les années objets sont bien derrière nous, l'approche Perrot est plus que jamais d'actualité pour aborder ce nouvel ordre des choses.*

ABSTRACT.

MOTS-CLÉS : *objets, règles, contraintes, multimédia, traitement du signal.*

KEYWORDS: *objects, rules, multimedia, signal processing.*

1 Introduction

La question « les objets ont-ils échoué ?» évoque à la fois la nécessité de faire un bilan, après plusieurs décennies de recherches dans le domaine des technologies à objets, et de proposer une perspective. Je tente ici de répondre à ces deux aspects de la question en soulignant le rôle déterminant des problèmes et des exemples de travail dans ces recherches.

2 Bilan : les années Objet

Ce bilan sur les années Objet est provoqué par le rassemblement autour de Jean-François Perrot (alias *jfp*), qui a animé et rendu possibles la plupart des réflexions sur les objets en France, leur donnant une impulsion et une force qu'elles n'auraient probablement pas eues sans lui. Ainsi, les « objets sous Perrot » sont avant tout des objets de discussions techniques dans lesquelles il est question de comprendre pleinement la nature et la portée des mécanismes, et en particulier les mécanismes de base de la programmation par objets (désormais POO) et de tous ses avatars.

Pas uniquement. Les objets auront aussi été un formidable prétexte pour parler de problèmes de représentation orthogonaux aux objets, voire au-delà : les structures (hiérarchiques, par points de vue), le temps, la classification, l'inférence. Plus que des objets d'études, les objets de la POO auront constitué un langage commun permettant de poser les problèmes typiques de représentation des connaissances de manière rigoureuse et intelligible. Enfin, les objets sous Perrot sont aussi un moyen de revisiter des pans entiers de notre culture partagée, de l'algèbre des types à la rhétorique antique[1], de la systématique à l'harmonie tonale.

Ainsi, au lieu de parler frontalement des objets (après tout, qu'en dire ? Ça marche !), je voudrais d'abord replanter le décor autour de ces questions de modélisation, dont certaines sont toujours d'actualité, pour ensuite proposer un bilan.

2.1 Le modèle de base : le point de vue Ikea

Rappelons le contexte par un cours sur la programmation par objets en 30 secondes, volontairement réducteur, et focalisé sur l'aspect domotique du mécanisme : sa capacité à *ranger*.

[1] A la question « que fait-on en TD tout à l'heure ? » je me suis entendu répondre « Eh bien aujourd'hui, vous faites la prosopopée de l'interprète ».

Grosso modo, la programmation par objets est fondée sur une « indirection bien placée »[2], qui fait basculer le monde de la programmation fonctionnelle (*f(x, y, z)*) à un monde qui le subsume, en permettant de ranger les fonctions dans un espace de noms propre à l'objet, ou plus précisément à sa classe : au lieu d'invoquer directement une fonction *f* supposée flotter dans un espace centralisé de fonctions - que le programmeur est censé connaître par cœur - on va localiser la fonction *f* (rebaptisée *méthode*) dans la classe de son premier argument (l'objet), passant ainsi, du point de vue de la notation, de *f (x, y, z)* à *x . f (y, z)*.

En donnant de l'importance à l'objet receveur plutôt qu'à la fonction, cet arrangement syntaxique permet bel et bien de ranger la fonction dans la classe au lieu de la laisser flotter en l'air. On crée alors un nouvel « espace de rangement » dans le sens où l'on remplace l'espace unique des noms de fonctions par un espace de classes, beaucoup plus grand car à tiroirs : grâce à l'indirection bien placée, chaque classe définit son propre espace de noms.

En outre, comme une cerise sur le gâteau, on peut se payer le luxe de ranger la classe *elle-même* dans un arbre d'héritage, et de donner sens à cet arbre en complétant simplement le mécanisme de recherche de la fonction (dans la classe de l'objet) par un mécanisme de « lookup » qui va chercher dans les super classes les méthodes non trouvées. Ce lookup s'inscrit miraculeusement sans heurt dans l'ensemble : l'héritage est en effet, dans sa version de base, complètement orthogonal à l'instanciation[3]. Dans une vision strictement opératoire et minimaliste, la seule différence avec la programmation fonctionnelle est donc que l'on peut démultiplier l'espace de noms des fonctions : en ayant de nombreux placards bien conçus pour ranger toutes ces fonctions, on peut du coup en écrire plus, plus correctement, les partager, et construire des programmes plus gros, plus sûrs, plus réutilisables, plus complexes. Grâce à ces deux petits tours techniques (indirection via la classe et *lookup*) on porte alors la programmation sur un terrain conceptuel jusqu'alors inexploré. Tout le reste découle de ce tour de passe-passe : le polymorphisme, l'encapsulation, la modélisation par objets en général.

Dans ce contexte, Smalltalk aura été un langage commun pour comprendre ces mécanismes en profondeur, mais aussi pour exprimer de nouveaux problèmes. Il n'aura pas été, contre nos espérances, un langage efficace pour capitaliser les développements. Certes, quelques systèmes ont pu être réutilisés, étendus, intégrés. Par exemple, les moteurs d'inférence (NéOpus, BackTalk) ou les systèmes d'interfaces graphiques (AckiaToolBox, HotDraw) furent utilisés et réutilisés. Mais point de capitalisation, de banque pérenne dans laquelle on eût pu véritablement puiser pour construire des systèmes complexes : l'enthousiasme initial de la réutilisation n'a pas été suivi de véritables effets. Mais la tâche était probablement

---

[2] L'expression est de jfp. Le reste n'engage que moi.

[3] Ce n'est plus vrai pour les langages réflexifs, et les liens entre instanciation et héritage ont fait l'objet de nombreux travaux par Pierre Cointe et Jean-Pierre Briot.

plus difficile que prévu. En tout cas, Smalltalk fut bel et bien un langage commun, le grec ou le latin de cette génération[4].

Si j'insère ici un rappel technique, c'est pour souligner l'importance de la coïncidence dans cette aventure des objets. Coïncidence entre d'une part une avancée technique (même réduite) et une avancée conceptuelle : penser en termes d'objets plutôt que de fonctions ouvre de nombreuses et nouvelles perspectives maintenant bien connues. Or dans le bilan parfois négatif que je propose ci-dessous, il me semble que ce qui aura manqué le plus souvent aura été non pas des propositions techniques ou conceptuelles, mais bien une coïncidence entre les deux : nous connaîtrons de nombreuses fausses joies liées à un défaut de coïncidence.

Le défaut de coïncidence peut être de niveau un : des avancées techniques sans résonance conceptuelle forte (par exemple les ATMS, dans un certain sens les contraintes) ou à l'inverse de niveau deux : des propositions conceptuelles séduisantes, mais sans substrat technique consistant (les agents, les aspects, les actes de langage, les méta-modèles, etc.).

2.2 La réification selon jfp

Réifier, ah que ce mot aura été utilisé ! Il s'agit cependant dans notre contexte d'un acte relativement bien défini : construire un objet informatique à partir d'un concept, voire conceptualiser un domaine de manière à produire ces objets informatiques. Que les objets du monde réel soient « déjà là » n'est cependant pas anodin (Cf. Section 3.2).

Si l'activité essentielle d'un programmateur par objets est la réification, celle-ci, avant de s'incarner dans une classe ou des objets divers, passe par le langage. Dans ce contexte, un intérêt marqué pour les langues, la littérature, et les mots en général, n'est pas accessoire[5], il serait même un pré-requis : bien réifier une situation, un problème, c'est avant tout trouver les mots pour le dire.

Or derrière cette apparente évidence ce cache sinon une hypothèse forte, du moins une croyance tenace, précisément en la puissance de la réification : l'idée qu'une fois le mot trouvé, et bientôt l'objet manquant identifié, puis conçu, décrit, voire inventé et enfin programmé, tout va mieux, nécessairement. L'idée encore que là se trouve l'essentiel de l'acte de modélisation, bien plus que dans la recherche

---

[4] Malgré la popularité du langage Java, on peut constater qu'il ne joue pas un tel rôle de lingua franca. Ceci est probablement dû à la médiocre qualité des classes de base, et au manque d'accès réflexif au langage et à son environnement.

[5] Chez jfp, cet intérêt pour la littérature – au sens propre, les lettres - va de pair avec un désintérêt, voire un certain mépris, pour la littérature « scientifique » ou du moins celle que l'on peut trouver dans les publications du domaine. Il faut dire que l'on peut trouver mieux que les proceedings d'Ecoop ou d'Oopsla comme livres de chevet.

d'algorithmes, de performances, qui bien souvent révèlerait plus un besoin de compétition (je vais plus vite que les autres, je suis plus malin, etc.), jugé impropre ou indigne, qu'un intérêt véritable pour l'essence du problème.

Cette attitude vis-à-vis de la réification considérée comme centrale est à la fois irrésistible et paradoxale. Irrésistible parce que bien souvent impossible à réfuter : comment lutter contre les mots, avec des mots, surtout contre jfp ? Paradoxale car l'informatique est avant tout l'art du calcul donc de la procédure. Ainsi un tel engouement pour le concept a-t-il quelque chose de bien singulier[6].

Ces mots, ces concepts réifiés sont à leur tour des moyens de faire des théories. Des théories sur tout ce qui se prête à la représentation, et qui permet de parler d'un réel partagé de manière raisonnable. Théories *sur* tout, mais pas théories *de* tout : cet engouement pour la modélisation se situe exactement à l'inverse des tentatives théoriques universalistes, ces théories de tout, dont il est question en physique pour unifier la gravitation avec les 3 forces élémentaires (Cf. Hawkins, 2002). Mais cette recherche du mot juste à la racine de la réification, elle-même censée sous-tendre une théorie s'accompagne aussi d'un souci de consistance : « Fuyons les théories qui n'ont point de théorèmes »[7]. Or cette lutte contre le vide n'est pas toujours consensuelle ni évidente : paradoxalement, les technologies objets peuvent avoir en effet le pouvoir de cacher des inconsistances.

Ainsi en est-il à mon sens, par exemple, des nombreuses tentatives de réifier les procédures et les programmes en général. Certes il est facile de créer une classe `Procedure`, de dire qu'une procédure a un nom, des types d'arguments, des pré ou post conditions. Mais au fond l'essence même de la procédure échappe à la réification (dans ce cas, on pourrait presque dire qu'elle échappe à la réification par construction, par l'idée même de l'ordinateur). Une des conséquences de ces pseudo réifications qui ne représentent que la surface de la chose est qu'il n'existe pas de théorie de l'égalité (comment savoir que deux fonctions sont équivalentes ?).

Ce débat récurrent sur la représentation des procédures semble infini. Cependant, avec Perrot la question n'est pas qu'ontologique (de la nature des procédures « en général »), mais doit être, pour être fructueuse, ancrée dans un certain usage, un terrain exemplaire qui précise la question, voire qui lui donne vraiment du sens. Représenter les procédures : pour quoi faire ? Ainsi, la programmation génétique (Koza, 1992) offre une représentation des procédures relativement convaincante, car associée à des processus (*cross-over*, mutation, etc.) opérationnels et dotés d'un contexte qui leur donne un sens incontestable (l'optimisation le plus souvent). Dans le contexte du typage (par exemple pour la compilation ou les systèmes distribués de

[6] J'oserais le rapprocher ici du mépris exprimé par le mathématicien René Thom à propos de l'équivalent de la procédure informatique pour les mathématiques, que pourrait être la démonstration : « il y aura toujours des … algébristes pour démontrer les théorèmes » (Kantor, 1999). Effectivement, l'informatique ne manque pas d'algorithmiciens.

[7] L'expression est citée dans (Perrot, 1996) à propos de Marcel-Paul Schützenberger.

type Corba), une procédure est parfaitement représentée par les types des arguments et de sortie. Dans l'environnement Smalltalk, les procédures (messages) sont réifiées dès qu'une erreur survient – dès que ceux-ci ne sont pas compris – et un *debugger* miraculeux s'ouvre alors. Dans tous ces cas, l'essence même de la procédure semble parfaitement représentée par le système, et coïncide avec l'usage que l'on veut en faire. Il me semble que l'on pourrait formuler l'hypothèse que les procédures ne se représentent pas bien … quand on ne sait pas quoi en faire.

On voit bien ici que la question du statut ontologique est en fait trompeuse : le statut ontologique ne se discutant que dans un certain contexte applicatif, celui-ci perd du même coup sa raison d'être : l'ontologie s'intéressant en principe à l'essence de l'être, celle-ci ne saurait en effet être contextuelle, du moins selon l'orthodoxie, sauf à recourir à des contorsions plus ou moins rationnelles.

Mais là est bien le fond du problème : la programmation par objets nous fait croire que l'on représente les choses d'une manière essentielle, alors que toute l'activité de modélisation nous montre que les objets n'ont de sens que pour une certaine classe d'application et d'usages. S'il fallait choisir entre une posture ontologique générale (traquer l'essence des êtres, vu d'en haut), et une position empirique, dans laquelle le sens des modèles est donné par le contexte – un sens qui s'évanouit dès que le contexte est supprimé voire simplement élargi - il me semble que l'école Perrot consiste précisément à se placer dans la deuxième position, et donc à préférer l'étude d'un réel ancré, tel qu'il est et qu'il résiste, à une contemplation idéaliste, trop vite universelle, dont le risque est de conduire à des positions de principe rigides et des dérives idéologiques (par exemple conduisant à décréter que les procédures sont à bannir irrévocablement, par principe, Cf. Carré et al, 1995).

2.3 Les exemples nourriciers

Cette recherche empirique, dont on peut tracer les origines du côté des sciences expérimentales (de Claude Bernard à François Jacob), engage à un rapport étroit avec le réel : pour inventer la glycogenèse du foie, il faut bien observer de véritables lapins en train d'uriner « pour de vrai », et ce, pendant de longues heures, de longs jours, voire des années. Dans ce contexte, je voudrais revenir sur l'importance que revêtent les exemples pour Perrot dans ces recherches. Pas n'importe quels exemples, mais les « exemples exemplaires » ou nourriciers : ces exemples particuliers qui ne valent pas tant pour eux-mêmes, mais en tant qu'ils laissent entrevoir un champ entier du réel qui pourrait se modéliser de la même manière. Ainsi, le *factoriel* inaugure la récursivité. Le Fibonacci par règles[8] symbolise les bases de règles d'ordre 1 ou du moins la vision combinatoire de ces règles, centrée sur les performances du moteur d'inférence. Ces exemples sont nourriciers car ils

[8] Dans sa version volontairement inefficace par 3 règles d'ordre 1 (`creeFils`, `sommeFils`, `fin`), inventée je crois par Christian Dujardin d'Ackia.

permettent d'identifier ce qui est important, difficile, mais donnent aussi des intuitions ou des idées. Ce sont des exemples sur lesquels la pensée peut s'accrocher pour abstraire ou inventer.

Pas seulement. Pour être véritablement nourriciers, ces exemples doivent aussi ouvrir sur de nouveaux champs. Ils doivent évoquer une infinité d'autres problèmes du même ordre, partageant de manière évidente avec l'exemple des traits essentiels.

Ces exemples sont tellement importants qu'ils peuvent prendre une place démesurée dans le discours, et faire croire à certains qu'on ne s'intéresse plus alors qu'à ceux-ci en tant que tels. Je voudrais préciser mon point de vue ici. Il ne s'agit pas de défendre une recherche appliquée par opposition à une recherche qui serait fondamentale mais exactement du contraire : c'est grâce à l'existence de ces exemples nourriciers que l'on peut aspirer à faire du fondamental. Pour faire du fondamental dans ce domaine de la réification, on est, il me semble, obligé de se confronter à des problèmes réalistes, par définition même de la notion d'ontologie, dans ce qu'elle a de plus fondamental, c'est à dire dans son rapport à l'essence du réel. Ainsi, les travaux de recherche en ingénierie des connaissances se réclamant d'une approche fondamentale cachent souvent, au contraire, des efforts pour se débarrasser d'un réel un peu trop encombrant et hirsute, pour se concentrer sur des *a priori* ontologiques qui n'ont, hors terrain exemplaire, rien de fondamental … ni d'appliqué.

2.4 Le ça marche

Enfin, ces exemples nourriciers sont aussi un procédé de validation expérimentale : le « ça marche ». Si le modèle permet de rendre compte de ces exemples, alors il est bon, solide, pertinent. Evidement ce « ça marche » a des limites qui sont précisément celles des exemples. Ca marche dans la mesure ou d'autres problèmes pourront se traiter de la même manière, sans nécessiter de torsion particulière, pour les faire coïncider artificiellement avec les modèles.

Dans le cadre de ce bilan, on peut aujourd'hui constater si « ça a marché » ou pas, en considérant les nombreuses extensions au modèle de base qui ont été tentées pendant ces années objet.

2.5 Les extensions du modèle de base ont-elles marché ?

Avec Perrot nous avons hérité d'une tradition constructiviste inaugurée par Patrick Greussay : pour comprendre un système rien de mieux que de le reconstruire en entier. Cette tradition s'est poursuivie avec Smalltalk, ou il s'est souvent agi de construire diverses extensions du modèle de base, pour étudier ces extensions dans un cadre naturel où l'on pouvait bénéficier de toute la panoplie d'applications

existantes, mais aussi pour mettre à l'épreuve le modèle objet de base, d'en voir les limites.

Ainsi de l'héritage multiple. L'héritage multiple tente de substituer un graphe acyclique à l'arbre d'héritage, mais rend alors le parcours de ce graphe possiblement ambigu. Certes, des solutions techniques plus ou moins raffinées ont été proposées (Ducournau et al. 95) mais hélas, le résultat coince, ne résiste pas au passage à l'échelle, ne coïncide pas (ou plus) : même si les problèmes d'ambiguïté peuvent être résolus dans certains contextes, les exemples réels doivent être passablement tordus pour être représentés par ce mécanisme : l'héritage multiple perd vite pied face à l'usage, aux exemples, au réel. Ca ne « marche pas ». Il en est de même pour les propositions d'instanciation multiple, ainsi que les tentatives non pas d'extension mais de simplification du modèle, par exemple les langages de prototypes qui ont tenté de supprimer les classes pour les remplacer par la délégation[9].

Du côté de l'extension vers la réflexion, de nombreux travaux seront consacrés à la représentation réflexive des objets, avec le modèle solaire ObjVlisp. Mais ici encore, il semble bien que les espoirs mis sur la réflexion structurelle (les métaclasses) et les techniques sophistiquées pour les représenter n'aient finalement pas coïncidé, non plus, avec de réels terrains exemplaires. D'une certaine manière, le modèle Smalltalk, bien que simplificateur, suffit pour exprimer tout ce que l'on voudrait décemment exprimer à un tel niveau méta.

Ainsi de même des usages limites du typage (covariance / contravariance) : même si certains modèles permettent en théorie de sauver la face, la contravariance ne *marche pas* : aucun programmeur ne peut construire de gros programmes en utilisant la contra-variance : si elle est logiquement fondée, la contravariance est conceptuellement contre-intuitive. Pas de redéfinition des types des arguments dans une méthode d'une sous classe. Dont acte.

Oserai-je émettre qu'il en est de même pour les fonctions génériques[10] ? Et pour un certain nombre de bizarreries introduites par Eiffel avec les meilleurs intentions du monde (par exemple la duplication de certains champs par l'héritage), mais qui, en fin de compte rendent le programmeur schizophrène.

De nombreux systèmes – plutôt qu'extensions - de représentation de connaissances ont aussi abordé des questions orthogonales aux objets : SysTalk (la thèse de Francis Wolinski, Perrot & Wolinski, 1992) la représentation multi-point de vue et hiérarchique (Cf. Figure 1) pour la modélisation de robots et plus généralement les extensions de Smalltalk pour l'intelligence artificielle et la résolution de problèmes : NéOpus, langage de règles d'ordre 1, basé sur le système Opus de Xerox (Cf. Pachet, 1995), BackTalk (les contraintes), EpiTalk (l'espionnage) et bien d'autres.

---

[9] Reportant alors tout ce manque sur la complexité souvent inextricable de la méthode « copy ».

[10] Très jolies du point de vue technique, mais très complexes à l'usage.

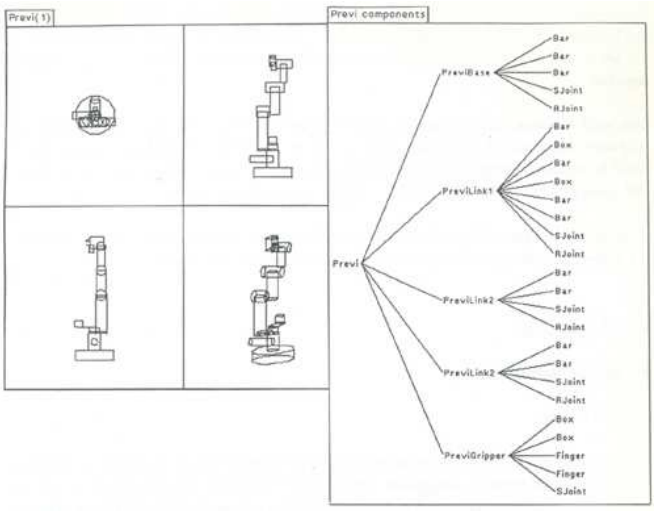

Figure 1. Le bras Prévi, exemple nourricier pour l'étude de la représentation multi-point de vue dans SysTalk (courtesy of Francis Wolinski).

Si ces tentatives sont souvent parties de bonnes intentions, elles se révèleront malheureusement aussi souvent de fausses bonnes idées, par défaut de coïncidence : soit de niveau 1, i.e. manquant d'exemples et d'ancrage dans le réel (la réflexion, la classification, les logiques de description et plus généralement les extensions basées sur de la juxtaposition multi-paradigme) soit au contraire de niveau 2, i.e. manquant de substance technique ou scientifique (les agents, les métamodèles, les sujets, les aspects, etc.).

Je voudrais expliquer mon propos en revenant sur certaines de ces extensions tentées par notre équipe dans les sections suivantes.

2.5.1 Sur les règles

Nous avons avec NéOpus étudié l'intégration d'un mécanisme de règles de production d'ordre un dans un langage objets, en l'occurrence Smalltalk. L'intérêt principal d'un tel choix était de bénéficier gratuitement de toutes les modélisations (entendre les instances de classes) réalisées en Smalltalk : on avait à notre disposition un champ apparemment infini de problèmes sur lesquels on allait pouvoir écrire des règles et travailler.

Mais il s'est assez vite avéré que de nombreux problèmes mettaient surtout en avant les aspects combinatoires, l'ordre 1. En effet, les applications types, les exemples nourriciers sont surtout des exemples calculatoires, avec en particulier le fameux Fibonacci, considéré comme l'exemple fétiche pour comparer les

performances des moteurs d'inférence de l'époque (NéOpus, Oks, LoopsIris, Essaim, etc.). Dans cette course à la vitesse, les astucieux mécanismes de compilation par démons de Voyer (1989) - qui ont donné lieu à la notion de *behavioural match* (Bouaud et Voyer, 1996) – n'ont d'ailleurs probablement pas eu le succès qu'ils méritaient.

Outre les problèmes de performance, un des aspects intéressants du travail sur les règles et qui nous concerne ici est justement le problème de réification qu'elles posent. La notion de règle est à la croisée d'une notion logique (assertion du premier ordre) et opératoire (la règle de production, sorte de procédure pendant dans le vide, jamais appelée explicitement). Or on a pu observer, avec le développement du système NéOpus, que la notion de règle ne prenait vraiment son sens qu'à partir du moment où était introduite la classe `RegleDeclenchable` (dans la syntaxe concaténative Smalltalkienne) et surtout sa méthode `declenche`, dont le sens est donné par le réseau Rete sous-jacent, qui, lui, ne bénéficia jamais d'une telle attention de la part de notre maître, probablement à cause de son caractère bassement algorithmique[11]. Les règles elles-mêmes n'étaient finalement que des méthodes factices, dans des classes elles-mêmes un peu factices représentant les bases de règles, des choses au fond peu importantes. Non que l'on n'ait pu réifier la règle *sui generis*, mais plutôt que la notion véritablement importante, intéressante était juste à côté. Seul un effort d'implémentation abouti permet de s'en rendre compte, *in vivo*.

Un des plus beaux exemples d'application de NéOpus a été le système NéoGanesh (Cf. Figure 2), objet de la thèse de Michel Dojat (Dojat et al, 1996). NéoGanesh est une application de contrôle de respirateur pour les malades en unités de soins intensifs. Son développement abordait essentiellement le problème de la représentation du temps, et des systèmes tournant en boucle fermée et en temps réel. Les travaux effectués par Michel Dojat avec notre équipe ont consisté à appliquer nos techniques au problème de la ventilation artificielle, avec des résultats parfois spectaculaires (Dojat et al. 1996), et une utilisation puissante des mécanismes propres à NéOpus (méta règles, héritage dit naturel dans les règles[12], etc.). Les tentatives pour transférer cette technologie vers un constructeur de ventilateur ont échoué à l'époque. Aujourd'hui, la ténacité de Michel Dojat a payé, et un tel système existe, intégré à un ventilateur du commerce (Mersmann & Dojat, 2004). Néanmoins, le retour sur investissement intellectuel est relativement faible. D'une part les techniques utilisées dans l'application finale sont bien pauvres : plus d'ordre 1 véritable, plus d'héritage « naturel », plus de méta-règles, pourtant mises en avant

[11] Attitude qui contrastait fort avec celle de Forgy et de Miranker, présents lors du workshop sur les EOOPS organisé à OOPSLA 91 (Pachet, 1994), et qui, eux, étaient effectivement préoccupés essentiellement par les questions de performance, et finalement peu intéressés par l'intégration d'objets dans leurs réseaux.

[12] Une sorte de transposition du mécanisme des fonctions génériques pour les règles d'ordre un, par lequel une règle peut filtrer des instances d'une classe et aussi de ses sous-classes.

dans nos présentations comme essentielles (Dojat & Pachet, 1992). D'autre part la filiation entre le produit et les travaux universitaires a pratiquement disparu.

On peut se poser la question de l'impact de NéOpus, 10 ans après, bien qu'un tel bilan soit difficile à faire, Smalltalk 2.5 n'étant plus guère utilisé. Une tentative de commercialisation a échoué mais quelques portages ont été réalisés. Les travaux sur NéOpus sont toujours cités, en particulier grâce à l'engouement récent pour la notion de règles métier (*business rule*), par exemple (D'Hondt et al, 2004) qui revisite ces questions d'intégration objets + règles dans le cadre de la notion, discutable à mon avis, d'aspect.

De manière plus neutre, un écho de l'impact peut être trouvé en effectuant quelques requêtes simples sur Google[13] et en regardant le nombre de pages rendues :

```
NéOpus : 476 pages.
NéOpus + pachet = 122 pages
```

Du point de vue de son impact et de la durabilité du projet, NéOpus aura donc été un relatif succès, relativement du moins aux projets similaires hexagonaux. Malheureusement, sa diffusion a été restreinte par manque de moyens, et il apparaît clairement aujourd'hui que nous aurions pu bénéficier d'un impact plus important, car les idées étaient bonnes et sont toujours d'actualité. Mais les exemples n'ont pas véritablement suivi.

Figure 2. Le système NéoGanesh en action. De vrais patients contrôlés par des méta-règles. Dans la version finale, les méta-règles ont disparu.

Au fond, ce mécanisme provoque aujourd'hui un sentiment *d'overkill*. Le mécanisme est puissant, certes mais les problèmes que l'on a à résoudre ne sont que partiellement d'ordre un. Il résistent, ils s'épuisent face à ce moteur efficace, mais

[13] La méthode est certes grossière mais intéressante car, précisément, aveugle.

qui tourne un peu à vide. Nous avons privilégié la voie en profondeur (inférences complexes, sur peu de règles) et il fallait probablement faire le contraire (raisonnements peu profonds, mais en largeur)[14]. Le présent (2004) semble nous donner raison : Patrick Albert (Albert, 2003) montre que si les règles reviennent en force dans le paysage technologique industriel, elles sont maintenant utilisées pour des problèmes bien différents : plus simples combinatoirement, mais plus larges. Hélas, les problèmes en question échappent au domaine universitaire[15].

2.5.2 Sur les contraintes

La programmation par contraintes (dont je trouve l'origine dans les problèmes combinatoires traités par le système Alice (Laurière, 1978) plutôt que dans la programmation logique) a eu du succès car elle a coïncidé à son origine avec une collection substantielle de problèmes combinatoires qui ont semblé ouvrir un territoire gigantesque[16]. Bien sûr des travaux en recherche opérationnelle cherchaient à apporter des solutions *ad hoc* à certains types de problèmes (linéaires par exemple), mais à l'époque le terrain des problèmes combinatoires « en général » n'existait pas en tant que tel[17].

Depuis les premiers systèmes de représentation de contraintes par des graphes (Alice), ou bien par les techniques dites de perturbation (Thinglab d'Alan Borning) les techniques ont certes beaucoup progressé[18], et ont été intégrées à des langages de programmation existants (d'abord Prolog, puis tous les autres). Si le domaine d'application des contraintes était prometteur au départ (les problèmes intrinsèquement combinatoires semblaient, par leur nature souvent ludiques, toucher un très large domaine), celui-ci s'est peu à peu spécialisé, réduit, et est sorti du champ universitaire, pour devenir la propriété des industries spécialisées.

Malheureusement, les problèmes combinatoires n'ont pas germé, et il s'est assez vite avéré que ceux-ci étaient circonscrits à un contexte industriel, voire à une communauté spécifique, et que l'élargissement de ces problèmes à des problèmes plus généraux ne se faisait pas. Faute de renouvellement des problèmes, on peut constater aujourd'hui l'épuisement intellectuel du domaine, et un repliement sur de la « théorie » qui fonctionne essentiellement en circuit fermé.

---

[14] C'est d'ailleurs la constatation faite par Douglas Lenat dans sa critique des systèmes experts et qui est à la base du projet Cyc.

[15] Comment modéliser une gestion de paie ou de comptabilité de manière réaliste dans un contexte universitaire ?

[16] Il fallait là un certain génie, celui des pionniers de l'IA et probablement de Laurière lui-même pour « réifier » le concept même de *problème combinatoire*. Idem pour GPS et la notion de « problème »…

[17] Il n'était pas réifié…

[18] L'extension de la littérature sur les contraintes ne correspond pas nécessairement à de réelles avancées. Jean-Louis Laurière pensait, en 1994, que « les contraintes ? le problème est réglé depuis longtemps » (communication personnelle).

J'ai proposé dans (Pachet, 1999) quelques nouveaux types de problèmes, en particulier le contrôle de spatialisation. Ces arguments sont repris en Section 3.

2.5.3 Sur la méta-modélisation

Comme nous avons tenté de le dire, la réification tout azimuts a des limites : il ne faut réifier que ce sur quoi on a quelque chose à dire, ou à faire. Pire, il est dangereux de réifier aveuglément, sans l'assise donnée par l'expérience et l'intuition que l'objet une fois produit sera utile « ontologiquement ». La métamodélisation à la MetaGen (Revault, 1995) repose sur un *a priori* important : qu'il est utile de réifier le méta-modèle, voire les méta-modèles source et destination du processus de transformation au cœur de la modélisation. Expérience faite, l'hypothèse est-elle une bonne idée ?

Selon moi, cette hypothèse n'est pas valide car il n'y a pas eu, ici encore, coïncidence entre une technique développée et un terrain exemplaire. En effet, le champ des problèmes à résoudre (à métamodéliser) s'est révélé inadapté à la technique : problèmes soit trop simples pour justifier un tel empilement conceptuel, soit trop complexes, trop mal posés, hétérogènes, et impossibles à résumer (par exemple, la modélisation effective d'un problème vraiment complexe comme ceux posés par la maintenance du réseau électrique d'EDF). Par ailleurs, les problèmes réels, concrets, de transformation de programmes ne sont pas résolus par cette approche. On ne sait toujours pas, par exemple, transformer correctement un programme Java 1.1 en Java 1.4.2, et ce problème ne relève pas, apparemment, d'un manque de méta-modèles.

L'hypothèse fondatrice de MetaGen n'est pas une hypothèse philosophique, que l'on pourrait démontrer ou justifier par des arguments ontologiques, mais elle relève d'une pétition de principe élaborée à partir d'intuitions accumulées lors de processus de modélisation concrets. Elle ne peut donc être validée ou invalidée que par l'expérience. Or l'expérience montre jusqu'à présent[19] que cet édifice ne permet pas de résoudre des problèmes que l'on ne pourrait pas résoudre avec des méthodes plus simples : forme particulière de non-coïncidence de niveau 1. Dont acte ?

[19] Après une dizaine d'années de pratique.

2.5.4 Sur les interfaces graphiques

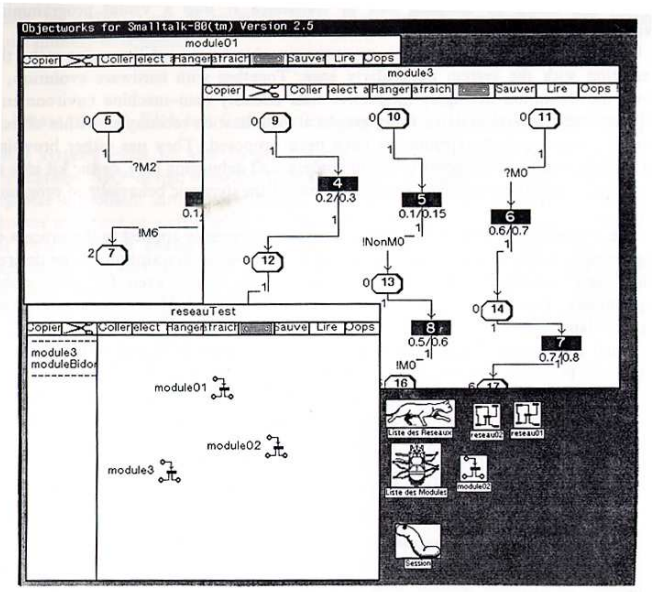

Figure 3. Une interface réalisée avec l'outil AckiaToolBox en Smalltalk. Ici, des réseaux de Petri.

Même si on peut retracer d'une certaine manière l'origine des objets dans la fabrication des interfaces graphiques[20] (notamment pour implémenter les visions du Dynabook d'Alan Kay), les objets ont, en retour, été beaucoup utilisés pour construire de nouvelles interfaces graphiques. Cette utilisation des objets, depuis MVC jusqu'aux générateurs d'interface modernes a tellement bien fonctionné, qu'à l'instar des moteurs à essence, ce sujet est sorti de l'université pour, simplement, se fondre dans les applications. Il y a eu coïncidence absolue, jusqu'à dissolution.

Qu'en dire ? Aujourd'hui probablement rien, mais je ressuscite un instant (Figure 3) l'*Ackia ToolBox*, cet ancêtre qui servit à construire de nombreuses interfaces, comme ces réseaux de Petri, à l'occasion d'un contrat avec le CSTB[21].

---

[20] En tout cas on peut dire que les widgets graphiques forment un terrain exemplaire … exemplaire.

[21] Ce fut à cette occasion que jfp s'exclama, devant nos piètres et minuscules exemples : « mais votre démo, c'est une contrepèterie ». Suite à quoi nous développâmes des exemples gigantesques, tellement qu'on ne pouvait plus les visualiser. Entre le jouet, trop petit et peu significatif et l'exemple complet, trop réel,

Aujourd'hui, le domaine des interfaces utilisateurs est devenu non seulement un domaine en soi, mais un terrain extrêmement actif. L'utilisation des objets a probablement culminé avec les générateurs d'interfaces réflexifs (*SOS interface* puis *InterfaceBuilder* de Jean-Marie Hullot). Il est probable que de nouvelles interfaces utilisateurs apparaissent dans le domaine de la multimodalité (son, mouvement, capteurs divers, Cf. section 3.2.3 pour un exemple).

2.5.5 Sur les langages de représentation

Sur le terrain dit de la « représentation » je pense qu'on peut dire qu'il y aura eu[22] échec à peu près complet. S'il est indéniable que le papier originel de Minsky (Minksy, 1995) aura suscité de nombreuses vocations langagières dans à peu près tous les pays du monde, on peut se demander aujourd'hui si l'enthousiasme suscité par la déclarativité n'était pas un peu aveugle, au vu des résultats.

Il me semble que le problème majeur posé par les langages dits déclaratifs, est qu'en refusant systématiquement l'usage de procédures ou de calculs quels qu'ils soient - considérés non suffisamment déclaratifs - on se coupe d'une bonne partie de ses munitions, de ses exemples nourriciers, de ses problèmes. Du point de vue technique ou « théorique », il me semble aussi, avec du recul, que l'opposition programmation / représentation (Carré et al., 1995) et le long débat qu'elle a entraîné n'aura *in fine* produit que peu d'éclaircissements sur les problèmes fondamentaux de représentation. En revanche, ce que je retiens de ces travaux dans le domaine de la représentation des connaissances, qu'ils soient considérés comme de la programmation ou de la représentation, c'est essentiellement la large palette des problèmes traités par les uns et les autres : des problèmes souvent originaux, allant puiser dans des disciplines éloignées de l'informatique (de la biologie à la musique), plus que les solutions apportées qui, pour la plupart, n'ont pas été généralisées.

Enfin en ce qui concerne les logiques de description, on est il me semble dans le cas typique[23] d'un défaut de coïncidence de niveau 1 : la technique est substantielle, mais on attend toujours un exemple convaincant[24].

2.6 La fin des nouveaux paradigmes

Ce petit bilan à caractère négatif coïncide lui-même avec un désintérêt massif de la communauté de recherche pour les (nouveaux) langages de programmation. En effet, on peut affirmer que depuis les objets, l'informatique n'a pas connu de

---

nous n'avions pas trouvé ce juste milieu qui permet de passer de la contrepétrie à la démo convaincante.

[22] Ce futur antérieur revêt ici un caractère diplomatique

[23] On pourrait dire ici nourricier, dans une perspective récursive dans laquelle on aurait à modéliser la théorie de la coïncidence elle-même.

[24] Cet exemple pourrait venir du web sémantique, mais il tarde à se manifester.

nouveau paradigme au sens fort où on l'entend ici, c'est à dire coïncidental. Par analogie avec le christianisme, considéré par (Gauchet, 1985) comme la religion de la fin des religions, le paradigme objet serait-il le paradigme de la fin des paradigmes ?

Possible. Si l'on considère les trois principaux paradigmes de programmation (fonctionnel, logique, objets), on peut constater que dans chaque cas, la naissance du paradigme s'est fondée sur une coïncidence forte entre des problèmes nouveaux et relativement bien définis (gestion de liste, moteurs d'inférence, transformations d'arbres, traduction, envoi de message, interfaces graphiques) et des techniques nouvelles et adaptées. Dans tous les cas aussi, la sémantique et la "théorie" sont venues après l'invention et la découverte. Dans tous les cas aussi une certaine urgence, et une apparition quasi instantanée des concepts...

Or, un peu comme le pétrole en ce début de XXI^{e} siècle, les réserves de problèmes s'épuisent, ou du moins disparaissent du champ académique pour se concentrer dans les institutions spécialisées, le plus souvent des entreprises privées : difficile, à l'université, de s'intéresser à des problèmes de représentation nouveaux originaux car ces problèmes sont désormais ailleurs, à la suite du déplacement des industries motrices, comme nous le verrons plus loin.

Le multi-agent considéré comme paradigme relève, a mon sens, d'un défaut de non-coïncidence caractérisé de niveau 2 (le concept d'émergence est séduisant mais son application à la programmation n'a pas donné lieu à une avancée technique notable) : Avec les agents, il n'est définitivement plus question de résoudre des problèmes « exemplaires » (ou du moins les résoudre mieux), mais, au mieux, de proposer des « guidelines » à d'hypothétiques concepteurs en mal de conseils[25]. Ce manque de problématique est aujourd'hui flagrant, et on assiste à l'essoufflement de ces idées, pour cause essentiellement de manque de problèmes nourricier. Plus de problème, plus de valeur ajoutée, et donc plus de paradigme.

Un réel à modéliser qui n'est plus *déjà là*, qu'il faut sans cesse tordre voire réinventer pour le rendre compatible avec des idéologies (non-coïncidence de niveau 1) ou des mécanismes jugés utiles par principe (non-coïncidence de niveau 2). Comment sortir de cette impasse ?

3 Perspective : la numérisation de la culture comme nouveau paysage

Tentons ici de proposer quelques pistes pour le futur, en défendant l'idée que de nouvelles formes de coïncidences sont à prévoir, sur de nouveaux terrains exemplaires. C'est en changeant le domaine d'activité que les problèmes reviennent,

[25] Les recherches dont le but est de proposer des guidelines pour la conception par objets ou par agents … par les autres, mériteraient une étude spécifique : d'ou vient ce désir si vif de conseiller ?

des problèmes difficiles, exemplaires au sens ou nous l'entendons ici, mais qui sont d'une nature un peu différente des problèmes usuels.

Ce nouveau contexte exemplaire est produit par le formidable changement de statut que vit l'informatique de ce début du XXIe siècle. Plus précisément, les problèmes dont nous avons traité jusque-là relevaient d'une problématisation du monde particulièrement avantageuse : les objets que nous avions à représenter ou à traiter étaient « déjà là », ou du moins, déjà fortement réifiés. Ainsi en allait-il des `Etudiants` et de leurs superclasses `Humain` ou `Mammifère`, des `TransactionsBancaires` et de leurs opérations canoniques, jusqu'aux objets techniques (les différents types de fenêtres et de menus). La provenance de notre intérêt pour ces divers objets est multiple, mais elle fait toujours écho à des applications bien particulières : celles de l'industrie du tertiaire ; banques, finance, systèmes d'information et gestion au sens large (Wolinski & Legrand, 2004).

En ce début de XXIe siècle, on peut constater que les industries motrices de l'informatique ne sont plus les industries traditionnelles du secteur tertiaire. L'industrie motrice principale est celle du loisir au sens large : cinéma, musique, jeux vidéo, dans une certaine mesure les télécoms. Ceci est avéré par les données macro-économiques, mais on peut en voir l'écho jusque dans la fréquentation des conférences en informatique (Siggraph 2004 : 30.000 personnes, Oopsla 2002 : 2500, Ijcai 2003: 2000).

L'implication de ce nouveau contexte pour notre problématique « objet » est profonde. Paradoxalement, ce nouveau paysage induit en effet une régression de nos objets d'étude, parce que les « nouveaux objets » ne relèvent plus d'une réification prédécoupée. Les nouvelles applications de l'informatique doivent désormais manipuler en temps réel vidéos, images, sons multi-canaux : les analyser, les synthétiser, le tout vite, et pour des applications exigeantes car pour le *grand public*. Or ces nouveaux objets sont très loin de se laisser faire aussi facilement que nos anciens `Etudiants`, `CompteEnBanque`, ou autres `Continuations` ou `MetaBaseDeRegles`.

Pour illustrer ce propos, et montrer ce qu'il a de positif voire d'exaltant, je propose des parallèles – des sortes de prolongements potentiels - des projets des années Perrot dans des projets actuels : du bras Prévi aux robots de loisir, de NéoGanesh aux systèmes temps réel multimédia, de BackTalk à la synthèse par contraintes, autant de nouveaux terrains exemplaires dont je suggère qu'ils recèlent les nouveaux exemples nourriciers de demain pour la recherche dans les langages de programmation / représentation.

3.1 Un changement de paysage

Le domaine du "multimédia", autrefois peu valorisé, a pris récemment une importance considérable. Voyez avec quelle ardeur notre société occidentale s'est mise à numériser son patrimoine audio-visuel !

Sans tomber dans les exagérations de la dénonciation systématique (le slogan de la « société du spectacle » de Guy Debord a suscité de véritables vocations contestataires chez les informaticiens (Codognet, 2000), allant parfois jusqu'à l'idéologie) on peut constater que l'*entertainment* a pris une place prépondérante dans notre société. Si les revers et les abus sont visibles (certains dénoncent à l'avance les dangers de l'accès à la culture dans son intégralité et le besoin de filtrer), il n'en reste pas moins que l'on peut préférer une société tournée vers l'*entertainment*, avec les travers que cela peut entraîner, à une société qui le dénigre systématiquement voire l'empêche ou l'interdit.

Ce qui nous intéresse ici est que cette évolution a de nombreuses conséquences pour notre champ d'étude, car elle modifie profondément la nature des besoins en applications : les problèmes nourriciers de l'informatique évoluent avec la société qui les a créés.

Ainsi, Philippe Breton dans son « histoire de l'informatique » (Breton, 1990) souligne le rapport fort qui existe entre les développements de l'informatique et les besoins de la société contemporaine. En particulier il souligne le rôle joué par l'armée américaine dans le développement crucial des calculateurs numériques dans les années 50. Ce développement permettra – de manière probablement non préméditée – l'essor formidable de l'informatique de la fin du XX^{e} siècle. Non seulement il s'agit là d'un rapport de force financier (les industries qui ont besoin de l'informatique financent les projets de recherche), mais aussi d'un rapport étroit de culture. En effet, les langages et les inventions informatiques entretiennent des relations profondes avec les projets qui les ont fait naître, y compris (voire surtout) pour leurs aspects les plus « théoriques » : lorsqu'on travaille sur la preuve de programmes, on a en tête une fusée qui explose (ou qui n'explose pas, si l'on est sûr de soi), une centrale nucléaire qui se dérègle (idem), un avion qui décolle avec plus ou moins de succès… Ces exemples ne sont pas, encore une fois, simplement des « applications » trouvées là par hasard, mais de véritables moteurs, des fournisseurs de ce réel dans lequel la pensée, même la plus inventive et originale, doit puiser pour se réaliser.

Plus tard, et jusqu'à la fin du XX^{e} siècle, ce sera essentiellement l'industrie des services qui sera motrice pour les développements de l'informatique : banques, finance, et l'informatique dite « de gestion ». Nous irons puiser dans ces cultures nos exemples de « transactions bancaires », de classifications du personnel (Personne / Etudiant, etc.), jusqu'à notre bien-aimé `Compteur` dont l'apparente abstraction (compter n'importe quoi) cache mal son origine dans des besoins, précisément, comptables. L'apparition momentanée de l'Intelligence Artificielle dans les années 80 fournira son lot de stéréotypes, dont on peut aussi retracer l'origine dans les applications des premiers systèmes experts : classifications de symptômes (Mycin) voire d'espèces (les Pingouins, les Mammifères). Aussi rudimentaires que soient ces exemples, ils sont encore une source vivante d'exemples et de matières à problèmes pour l'intelligence artificielle d'aujourd'hui.

Mais ce paysage a bien changé, par le rôle nouveau de la culture au sens large. Jusqu'ici reléguée à la marge de l'industrie, elle occupe maintenant une place croissante, à tel point que l'on peut assister à un changement de notre culture informatique elle-même. Par le terme *entertainment* nous désignons ici les industries du loisir, individuel ou collectif et essentiellement numériques (les jeux vidéos, mais aussi les films, la télévision, la musique, la photographie, et l'électronique grand public en général). Mais *l'entertainment* – au sens où nous l'entendons ici - n'est pas qu'une question de loisir, et désigne aussi « l'industrie de la culture », qui consiste précisément à rendre accessible la partie numérisable de nos patrimoines divers (musique, films, etc.).

3.2 Des objets d'une nature différente

Or la principale différence entre l'industrie de l'*entertainment* et les industries traditionnelles de l'informatique est dans la nature des objets qui y sont manipulés. D'une manière peut-être paradoxale, l'*entertainment* opère un sérieux retour vers le réel en manipulant des informations qui sont bien plus proches de notre monde physique, plus immédiatement sensibles, que les abstractions polies de l'industrie des services. Notamment, les signaux en provenance de capteurs divers (son, vidéo, image) mais aussi en sortie (synthèse, contrôle) y ont une importance prépondérante. Ainsi, si la dichotomie symbolique / numérique est ancienne, elle revient sur le devant de la scène sur des bases un peu différentes : plus que jamais, le symbolique et le numérique doivent coexister et sont aujourd'hui naturellement complémentaires dans nos nouvelles applications.

Pour illustrer ce propos, je propose ici trois exemples concrets de problèmes posés par l'industrie du loisir, choisis pour leur relation avec trois de nos anciens projets : la robotique de loisir, la spatialisation musicale et l'accès par le contenu aux collections de musique.

3.2.1 La robotique de loisir

La « robotique de loisir » est un nouveau domaine issu du croisement entre la robotique traditionnelle et l'industrie du loisir. Les applications visées dans ce domaine ne relèvent pas de la résolution de problèmes ou de l'assistance à des tâches spécifiques, mais plutôt la construction d'entités autonomes ayant des capacités de présence, d'attraction, d'attachement, et qui sont capables de s'insérer harmonieusement dans un contexte grand public, voire familial. Les exemples de tels robots se sont multipliés récemment sous diverses formes (voir un exemple à la Figure 4).

L'émergence de ce concept justifierait une analyse sociologique à part entière. Ce qui nous intéresse ici est plutôt la gamme de problèmes techniques que ces nouveaux objets posent pour le concepteur d'applications et pour le programmeur. En effet, si ces robots doivent répondre à des kyrielles d'exigences plus ou moins

traditionnelles (percevoir le monde, se mouvoir, se recharger, etc.) ils doivent le faire – du moins est-ce un des objectifs poursuivis - en l'absence de contexte téléologique prédéfini (tâche particulière comme l'identification ou la reconnaissance), voire être capables de créer à la volée des tâches en fonction des interactions avec leur environnement. En somme, ces robots doivent obéir à une seule contrainte forte : être intéressants pour ceux qui les regardent, ou interagissent avec eux.

De nombreux nouveaux problèmes se posent alors pour l'informaticien : comment construire des dialogues *intéressants*, comment doter ces robots de systèmes de perception à la fois *ancrés* et pouvant passer à l'échelle (*scalable*), comment les rendre curieux de leur environnement sans modèle prédéfini de celui-ci (Oudeyer & Kaplan, 2004), voire comment les rendre capables de s'intéresser par eux-mêmes à leurs activités, et rendre celles-ci autotéliques (Steels, 2004). Pour peu que l'on accepte cette idée de robots inutiles mais attirants, les problèmes techniques posés sont alors naturellement coïncidants, car ils entrent en résonance avec ce que l'humain a de plus mystérieusement utile, sa capacité à se développer.

Figure 4. Le robot Qrio est-il le Prévi de demain ?

3.2.2 Contraintes et temps-réel

Au-delà des règles, des contraintes, de la classification, etc, reste-t'il encore de nouvelles grandes classes de problème ? Il me semble que non. En revanche, il existe aujourd'hui de nouveaux contextes d'utilisation de ces techniques, qui en retour peuvent poser de nouveaux problèmes de fond. Je donnerai ici un exemple d'utilisation des techniques de contraintes dans le domaine de la musique : le contrôle de la spatialisation.

Dans le système MusicSpace (Pachet & Delerue, 2000), l'utilisateur peut modifier la spatialisation de sources sonores (par exemple les différents instruments d'un morceau de musique) de manière interactive. Mais cette liberté donnée à l'utilisateur engendre aussi la possibilité de produire des mixages incohérents, car l'utilisateur n'a pas les connaissances requises pour mixer correctement. Pour assurer que les mixages sont « corrects », une partie du savoir-faire de l'ingénieur du son est représenté dans MusicSpace sous la forme de contraintes (les petites boules, Figure 5). Ces contraintes agissent de manière réactive, par un algorithme *ad hoc* dans la lignée du système ThingLab. Ainsi les utilisateurs peuvent réaliser ce vieux fantasme de la manipulation interactive de sources tout en produisant des mixages raisonnables : la coïncidence est ici il me semble parfaite.

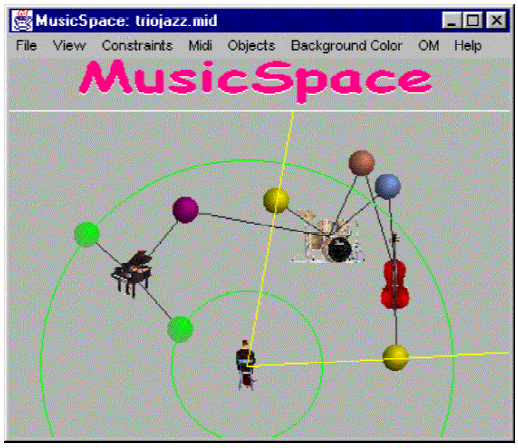

Figure 5. MusicSpace, un système de contrôle de la spatialisation de sources sonores à base de contraintes. Le multimédia, un nouveau terrain pour les solveurs de contraintes ?

Outre l'originalité de l'approche pour le mixage, l'intérêt d'un tel projet, à mon sens, est que la mise à l'épreuve du système dans des conditions réalistes (remixage interactif et écoute de musique populaire) fait en retour apparaître de nouvelles classes de problèmes et de possibilités : en donnant à l'utilisateur plus de contrôle sur le mixage, c'est l'idée même du mixage stéréo (figé par le support) qui est remise en question. Peut-on pousser plus loin cette idée, et imaginer par exemple que la musique serait systématiquement remixée en temps-réel, à la volée, permettant alors de nombreuses manipulations inédites (inouïes, au sens propre) ?

Le standard Mpg4 tente précisément de permettre la représentation de scènes audiovisuelles (films, sons) de manière structurée (Ronfard, 2004). Cependant, les liens entre les chercheurs en traitement du signal (Mpeg) et les spécialistes de représentation des connaissances sont aussi féconds qu'inexplorés. Le multimédia est

un domaine riche de problèmes exemplaires pour les techniques de résolution de problèmes (Pachet, 1999) : bien d'autres terrains d'expérimentation pour ces techniques sont possibles en dehors du terrain classique des problèmes combinatoires : *musaicing* ou la concaténation de sons à l'aide de contraintes pour la composition de musique électronique (Zils & Pachet, 2001) ou génération de *playlist* automatiques pour les systèmes d'accès à la musique en ligne (Pachet et al., 2000).

3.2.3 La gestion de grosses collections audio-visuelles

Un des aspects frappants de la numérisation de la culture, grâce aux progrès des techniques de compression (Mpeg) est la disponibilité pour le grand public de quantités gigantesques de documents audio-visuels. Le premier domaine touché par ce phénomène est la musique, mais les autres pans du secteur du multimédia (films, textes) sont déjà aussi atteints par ce phénomène. Pour ce qui concerne la musique occidentale « officielle », ce sont environ 10 millions de titres qui sont progressivement mis à disposition du plus grand nombre. La coïncidence est massive.

Or, si la mise à disposition de la musique a d'abord posé des problèmes légaux, elle pose maintenant des problèmes de représentation. D'une part, comment décrire la musique, pour construire des systèmes de recherche efficaces, véritablement utilisables ? Une des voies activement explorées aujourd'hui est la voie dite des métadonnées, nom donné aux descriptions de toutes sortes que l'on peut vouloir associer au simple et rudimentaire fichier audio pour en enrichir l'expressivité. Outre les questions de standardisation (Mpg7, Ronfard, 2004), la production – manuelle ou automatique - de ces métadonnées pose de nombreux problèmes de taille aux informaticiens : des problèmes d'ontologie classiques mais posés à grande échelle (comment classer les styles musicaux, sachant que toute classification n'est intelligible que localement, que de nouveaux styles paraissent sans cesse, etc); des problèmes perceptifs (comment extraire automatiquement des descripteurs acoustiques tels que le tempo, l'énergie, le timbre, etc.) ; mais aussi des problèmes de bases de données (comment construire une matrice de similarité pour 10 millions de titres ?).

Plus généralement, l'accès à la musique est, il me semble, représentatif de ces nouveaux problèmes mal définis. Pourquoi cherche-t-on de la musique ? Certainement pas pour résoudre une tâche particulière. Ces problèmes nécessitent de meilleures connaissances sur les aspects les moins rationnels du comportement humain (la recherche du plaisir, l'excitation par exemple), mais aujourd'hui les plus mis en valeur.

Figure 6. Du Browser Smalltalk au Music Browser : le MusicBrowser permet de trouver et d'écouter de la musique par des descripteurs perceptifs dans de grosses collections (10,000 titres). Ici, avec une interface à base de reconnaissance de mouvement via une webcam (sur l'image, pour sélectionner des titres par métadonnées (pays, genre, artiste) et changer le volume). Demain ces interfaces seront ubiques. La souris, les menus : souvenirs, souvenirs ?

La Figure 6 montre le système MusicBrowser développé à Sony CSL (Pachet et al., 2004). Ce système réalise une chaîne complète de traitement, de l'analyse automatique des fichiers audio pour extraire des descripteurs perceptifs (timbre, énergie, présence de voix) à l'analyse de similarités culturelles (par fouille de données sur le web).

Or pour construire cette application – et manière générale les applications de browsing multimédia par le contenu – un contrôle très fin doit être établi entre les différents niveaux de représentation, du signal audio pur aux taxonomies multiples de genre. Pour ce faire, le découpage traditionnel des langages de programmation (chaque type de problème à son langage) impose de manipuler de nombreuses couches logicielles parfaitement étanches : Matlab pour les algorithmes de traitement du signal, C++ pour leur réimplémentation efficace, Java pour les interfaces, Php/MySql pour les bases de données, etc. Nous sommes encore loin de l'homogénéité du système Smalltalk, qui représente à la fois les entiers, le compilateur et les applications dans le même environnement.

3.3 De nouveaux enjeux langagiers

Les trois projets esquissés ci-dessus tentent de construire des prolongations virtuelles de nos anciens projets, dans ce nouveau contexte du multimédia. Ces projets sont choisis comme représentatifs des nouveaux terrains exemplaires que je tente de décrire ici car ils permettent de poser certaines questions fondamentales de représentation et de programmation avec une approche expérimentale et constructiviste. Ces projets sont aussi l'occasion de réfléchir sur les nouveaux enjeux langagiers, au-delà des objets. J'identifie ici trois de ces enjeux posés à l'informaticien et le constructeur de langages de programmation / représentation, et pour lesquels la programmation par objets, si elle est toujours aussi indispensable, montre clairement ses limites.

3.3.1 Des langages de description de contenus

Un problème récurrent posé par ces applications est celui de la *description de contenus* numérisés. Ces contenus peuvent être à la fois statiques (collection de musique, de films, de livres, etc.), mais aussi dynamiques (le son d'une voix capté en temps réel, le flux vidéo d'une caméra, etc.). Les descripteurs dont on a besoin sont souvent appelés *perceptifs* car ils doivent, pour être utiles, coller au plus près à la manière dont un être humain décrirait ou catégoriserait ces signaux. Les travaux sur les descripteurs perceptifs font donc à la fois appel à des connaissances en traitement du signal mais aussi en psychologie et en cognition (de la vision, de l'écoute). Par ailleurs ces descripteurs sont souvent de nature intrinsèquement imprécise (le tempo par exemple, ou le timbre d'un son, d'un morceau). Certains standards comme Mpeg-7 se sont focalisés récemment sur le problème de la représentation de ces informations, mais laissent de coté le problème, bien plus difficile, de l'analyse et de la synthèse de ces descripteurs. Ces problèmes sont encore traités de manière artisanale, au cas par cas, et on ne sait pas capitaliser ces expertises, les combiner, les généraliser. Le projet EDS (*Extractor Discovery System*) conduit à Sony CSL est un premier pas dans cette direction, car il tente d'abstraire le processus de conception d'un extracteur (audio pour l'instant) dans son ensemble, en partant uniquement d'exemples (Zils, 2004). Un premier pas seulement car EDS reste un système relativement fermé, et il reste du travail pour en faire un véritable langage de spécification.

Ces descripteurs peuvent être vus comme les variables d'instance de Smalltalk rendues dynamiques, ancrées dans le réel modélisé (*grounded*), voire organiques. Un exemple nourricier est donné par la recherche de séquences vidéo dans une base d'archives personnelle telle que l'ensemble d'une vie filmée comme le pronostique le projet *MyLifeBits* (Gemmell et al., 2002)[26]: l'utilisateur pourra définir un descripteur personnalisé, par exemple « ambiance marine » en donnant quelques

[26] L'ensemble d'une vie filmée, image et son, tiendrait dans 500 téra-octets, soit la capacité d'un disque dur du commerce dans quelques années, Cf. (Delahaye, 2003). Il faudra bien alors indexer tout ça de manière intelligente ...

exemples de vidéos filmées à la mer. Il voudra ensuite trouver toutes les occurrences d' « ambiance marine » dans ses archives, et retrouver les films de ses vacances d'adolescent au bord de la mer… ou bien trouver les séquences avec un bruit de « cheval » dans sa vidéothèque. Le système, capable de trouver des descripteurs adéquats pour cette requête, éventuellement conçus par une communauté de chercheurs, lui dirait alors « un bruit de cheval ? Que voulez-vous dire exactement : un cheval au galop » ? au trot ? marchant au pas sur du gravier ? sous la pluie ? Ou autre chose encore ?».

3.3.2 Le dehors et le dedans : pour des langages verticaux

Les langages actuels imposent une dichotomie forte entre signaux et représentations symboliques. Or nos nouvelles applications ont besoin à la fois de manipuler des descriptions symboliques des contenus à l'usage des utilisateurs, mais aussi de controler finement, voire en temps réel, la manière dont ces descripteurs sont construits. Aujourd'hui, de telles dépendances sont techniquement coûteuses à réaliser car elles nécessitent de faire passer une grande quantité d'information à travers toutes ces couches logicielles, violant au passage le principe d'encapsulation.

En effet la recherche sur la description de contenu a été découpée jusqu'à présent en deux domaines étanches : d'une part le traitement du signal et la reconnaissance des formes, d'autre part les manipulations symboliques et l'ingénierie ontologique. Ici l'on a besoin des deux cotés du manche en même temps car la séparation n'est plus tenable. On voudrait ainsi un langage qui puisse définir la manière dont on veut extraire un descripteur à partir d'un signal, et ce de manière abstraite (pas du code Matlab !), et controler ce processus depuis l'interface utilisateur, à la volée : d'une certaine manière prolonger les racines de Smalltalk (qui s'arrêtent aux littéraux : entiers, chaînes de caractères) vers le bas (ou plutôt ici le dehors), tout en gardant la possibilité d'un contrôle de haut niveau sur ces structures. Un exemple nourricier est donné par le karaoké moderne, intégré à la chaîne Hifi du salon : un utilisateur choisit un titre de musique au hasard et chante en même temps ; sa voix remplace la voix originale, ce qui suppose détection, séparation, remixage, et intégration du tout dans l'interface utilisateur. On peut aisément généraliser l'exemple avec le clip vidéo dont le visage de la star en action est remplacé par celui de l'utilisateur…

3.3.3 Réifier et Manipuler de grosses collections

La numérisation de la culture crée de très grosses quantités de contenus : 10 millions de titres de musique occidentale, 60 milliards de photographies prises chaque année[27], etc. On retrouve ici le problème de la *réification des collections*, traité par tous les langages à objets modernes, mais qu'il faut maintenant élargir pour considérer des collections de nature potentiellement infinie, mal délimitées, et ne tenant pas en mémoire. Le *peer-to-peer* complique encore la gestion de ces bases, qui doivent aujourd'hui aussi facilement partageables, accessibles, tout en restant

[27] Source Kodak.

très efficaces. On voudrait ici, par exemple, manipuler des informations sur de grosses collections qui ne sont pas nécessairement organisées en tant que telles, comme le web ou une collection de fichiers vidéo traînant[28] sur des disques durs en réseau, et considérée, par le programme, comme une collection à part entière. Or les travaux sur les bases de données objet n'ont pas réussi à marier efficacement (du point de vue conceptuel du moins) bases de données et langages de programmation. Par ailleurs, le calcul de description sur de grosses bases pose des problèmes d'efficacité redoutables : calculer la matrice de similarité entre tous les titres des titres de musique d'un utilisateur (typiquement 10000 titres) demande environ 10^8 calculs de similarité, ce qui est hors de portée des machines actuelles. Un exemple nourricier est ici le problème du calcul de la similarité entre tous les titres du catalogue d'un gros label de musique (500 000 titres), ce qui permettrait d'enrichir la représentation en machine du catalogue avec des propriétés importantes comme sa densité, sa connexité, sa structure : bref, de le réifier correctement !

4 Conclusion

Les chercheurs qui ont connu le début de l'époque des objets (la fin des années 70, le début des années 80) ont eu de la chance. Ils ont pu participer activement à la naissance d'un nouveau paradigme. Depuis, malgré de nombreux essais et idées originales, la recherche dans les langages de programmation semble ne plus connaître de renouveau d'une telle ampleur, et les débats actuels (aspects, *patterns*, langages de modélisation, méta-modélisation, agents, standardisation) paraissent bien ternes en regard de l'excitation intellectuelle d'autrefois : manque de coïncidence entre avancées techniques et avancées conceptuelles, qui trouve souvent sa cause dans l'épuisement des exemples et des terrains nourriciers.

Cependant, l'expansion de l'industrie de *l'entertainment* crée de nouveaux domaines d'applications et aussi de nouvelles classes de problèmes. Techniquement ceci est lourd de conséquences : ces applications nécessitent de manipuler de manière homogène signaux et représentations symboliques, dans des collections de grande taille, et à l'usage du grand public, c'est à dire de non experts.

Si le paysage a changé en profondeur, l'approche Perrot est cependant toujours d'actualité : il faut trouver de nouveaux exemples nourriciers et les étudier avec un goût du réel préliminaire à toute théorisation. Et nos nouveaux exemples semblent se présenter en abondance, pour peu que l'on veuille bien admettre ce nouvel état des choses. Il est alors fort possible qu'après 20 ans de stabilité paradigmatique, on soit à nouveau dans la même situation que McCarthy, Colmerauer ou Kay, c'est à dire devant l'éclosion d'un paysage de nouvelles applications, l'apparition de nouveaux

[28] Le verbe *trainer*, bien qu'un peu péjoratif, est employé sciemment pour décrire le caractère instable, pas toujours officiel ni intentionnel de ces fichiers sur lesquels notre attention doit maintenant se focaliser.

terrains exemplaires, de nouvelles coïncidences, et donc à la veille de la création d'un nouveau paradigme, mais qui reste encore à inventer…

Bibliographie

Albert, P. Braunschweig, A. (2003) *Agents et services : deux (web) sémantiques*, Plateforme Afia, Laval, juillet.

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

Breton, Philippe (1990) Une histoire de l'Informatique, Seuil.

Carré, B. Ducournau, R. Euzenat, J. Napoli, A. Rechenmann, F. (1995) *Classification et Objets : Programmation ou Représentation* ? Actes des Journées du PRC-GDR Intelligence Artificielle, pp. 213-237, Nancy, Février.

Codognet, P. (2000) Mémoires virtuelles. Introduction au débat organisé au Musée d'Art Moderne de la Ville de Paris durant l'exposition "Voilà", Septembre 2000. http://pauillac.inria.fr/~codognet/art/voila.html

D'Hondt, M., Gybels, K. and Jonckers, V. (2004) *Seamless Integration of Rule-Based Knowledge and Object-Oriented Functionality with Linguistic Symbiosis.* In Proceedings of the 19th Annual ACM Symposium on Applied Computing (SAC 2004), Special Track on Object-Oriented Programming, Languages and Systems, Nicosia, Cyprus, ACM, March.

Debord, G. (1996) La société du spectacle, Folio Gallimard (réédition).

Delahaye, Jean-Paul (2003) "La mémoire, le calcul, l'intelligence", Conférence invitée, plateforme Afia, laval, juillet 2003. http://www.afia.polytechnique.fr/plateforme-2003/index.html.

Dojat, M., Harf, A., Touchard, D., Laforest, M., Lemaire, F., and Brochard, L. (1996) Evaluation of a knowledge-based system providing ventilatory management and decision for extubation, Am. J. Respir. Crit. Care Med, 153, 997-1004.

Dojat, M., and Pachet, F. (1992) Representation of a Medical Expertise Using the Smalltalk environment: putting a prototype to work, TOOLS Europe '7, G. Heeg, B. Magnusson, and B. Meyer, eds., Prentice-Hall, Dortmund (Germany), pp. 379-389.

Ducournau, R. Habib, M. Huchard, M. Mugnier, M-L. and A. Napoli. (1995) Le point sur l'héritage multiple. *Technique et Science Informatiques*, 14(3):309-345.

Gauchet, Marcel. (1985) Le désenchantement du monde. Gallimard.

Gemmell, Jim, Bell, Gordon, Lueder, Roger, Drucker, Steven, and Wong, Curtis (2002) MyLifeBits: Fulfilling the Memex Vision, ACM Multimedia '02, December 1-6, 2002, Juan-les-Pins, France, pp. 235-238.

Hawking, S. (2002) The Theory of Everything: The Origin and Fate of the Universe. New Millennium Press.

Kantor, J.-M. (1999) Communication personnelle.

Koza, John R. (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection, Cambridge, MA: The MIT Press.

Laurière J-L. (1978) A Language and a Program for Stating and Solving Combinatorial Problems, Artificial Intelligence 10, pp. 29–127.

Maisonneuve, S. Hennion, A. Gomart, E. (2000) Figures de l'amateur, La Documentation française/Ministère de la Culture.

Mersmann, S. and Dojat, M. (2004) SmartCare: Automated clinical guidelines in critical care, ECAI'04, Valencia (ES), August.

Minsky, M. (1975) A Framework for Representing Knowledge, The Psychology of Computer Vision, P. H. Winston (ed.), McGraw-Hill.

Oudeyer, P-Y. and Kaplan, F. (2004) Intelligent adaptive curiosity: a source of self-development, Proceedings of the 4th International Workshop on Epigenetic Robotics.

Pachet, F. (1995) On the Embeddability of Production Rules in Object-Oriented Languages, JOOP 8(4): 19-24 (1995).

Pachet, F. (1994) Report on the OOPSLA'87 Workshop on EOOPS. Addendum to the OOPSLA'94 proceedings. ACM SIGPLAN notice, http://www-poleia.lip6.fr/~fdp/eoops.html

Pachet, F. (1997) Langages à objets et représentation de connaissances, Mémoire d'habilitation, Université Paris 6.

Pachet, F. (1999) Constraints for Multimedia Applications, Proceedings of PACLP 1999, The Practical Application Company, London, March.

Pachet, F., Roy, P. and Cazaly, D. (2000) A Combinatorial Approach to Content-Based Music Selection. IEEE Multimedia, 7(1):44-51 March.

Pachet, F. Laburthe, A., Aucouturier, J.-J. (2003) Popular Music Access: The Sony Music Browser, Journal of American Society for Information Science.

Pachet, F. and Delerue, O. (2000) On-The-Fly Multi-Track Mixing, Proceedings of AES 109th Convention, Los Angeles, Audio Engineering Society.

Perrot, J.-F. (1996) Marcel-Paul Schützenberger (1920 – 1996), Revue du CNRS, Hermes, 20, pp. 259-261. Disponible aussi sur la page web du séminaire Lotharingien de Combinatoire : http://www.mat.univie.ac.at/~slc/

Perrot, J.-F. (1994) Objets, classes, héritage : définitions, *Langages et Modèle à Objets, Etat des recherches et perspectives*, INRIA, Collection Didactique, Ducournau, R. Euzenat, J. Masini, G. Napoli, A. Eds.

Perrot, J.-F. Wolinski, F. (1992) Modélisation par objets en robotique, Technique et science informatiques, Hermès, Paris, 11(1), pp. 97-115.

Revault, N., Sahraoui H.A., Blain G., Perrot J.-F., (1995) A Metamodeling Technique: the MetaGen System, in proc. TOOLS Europe'95 proceedings, TOOLS 16, Prentice Hall.

Ronfard, R. (2004) MPEG, une norme pour la compression, la structuration et la description du son, in *Informatique musicale - du signal au signe musical*, Pachet, F & Briot, J.-P. Eds, Hermes, pp.403-422.

Steels, L. (2004) The Autotelic Principle, In Fumiya, I., R. Pfeifer, L. Steels, K. Kunyoshi, editor, Embodied AI., Lecture Notes in AI (vol. 3139), Springer Verlag.

Striegnitz, J., Davis, K. (2003) Draft Proceedings of the workshop on Declarative Programming in the Context of Object-Oriented Languages (DP-COOl' 03). http://www.fz-juelich.de/zam/files/docs/ib/ib-03/ib-2003-11.pdf

Voyer, R. (1989) Implémentation d'architectures efficaces pour la représentation des connaissances : application aux langages Loopsiris et OKS, Thèse de l'Université Paris 6.

Wolinski, F. Legrand, J.-P (2004) L'objet tertiaire à travers les ages, Même volume.

Zils, A. and Pachet, F (2001) Musical Mosaicing, Proceedings of Digital Audio Effects Conference, DAFX'01, December, University of Limerick.

Zils, A. (2004) Extraction et exploitation de descripteurs musicaux perceptifs, Thèse de l'université de Paris 6.