Apport de differents paradigmes de programmation comme autant d'outils de pensee

Jean-François Cloutier

▶ **To cite this version:**

Jean-François Cloutier. Apport de differents paradigmes de programmation comme autant d'outils de pensee. Colloque francophone sur la didactique de l'informatique, Sep 1988, Paris, France. pp.195-203. edutice-00362448

HAL Id: edutice-00362448
https://edutice.hal.science/edutice-00362448

Submitted on 18 Feb 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

APPORT DE DIFFERENTS PARADIGMES DE PROGRAMMATION COMME AUTANT D'OUTILS DE PENSEE

Jean-François Cloutier

Ingénieur des Logiciels d'Intelligence Artificielle
Systems Designers

RÉSUMÉ

Chaque science propose son paradigme explicatif qui est idéalement complémentaire à ceux des autres sciences. Le progrès des sciences autant que le progrès intellectuel d'un individu semblent marqués par des transitions d'un paradigme explicatif dépassé à un nouveau paradigme plus approprié. L'utilisation réfléchie de langages de programmation, en tant qu'activité de développement intellectuel, doit tenir compte du fait que chaque langage, en règle générale, ne promeut l'emploi que d'un seul paradigme de représentation. Puisque la capacité de passer d'un paradigme à un autre semble cruciale au progrès intellectuel, il est contre-indiqué de confiner les activités de programmation à un seul paradigme. Chaque paradigme de programmation supporte un ensemble particulier de stratégies de résolution de problèmes. L'emploi de plus d'un paradigme de programmation offre un éventail plus riche de stratégies de résolution de problèmes directement applicables.

APPORT DE DIFFERENTS PARADIGMES DE PROGRAMMATION COMME AUTANT D'OUTILS DE PENSEE

Jean-François Cloutier

Ingénieur des Logiciels d'Intelligence Artificielle
Systems Designers

1. LE CONCEPT DE PARADIGME

Un paradigme est un point de vue particulier sur la réalité, un angle d'attaque privilégié sur une classe de problèmes, un état d'esprit etc.

Un physicien et un psychologue ne s'intéressent généralement pas aux mêmes types de problèmes, et lorsque c'est le cas, leurs points de vue respectifs sont radicalement différents. La physique, en tant que science, propose un vocabulaire descriptif en grande partie disjoint du vocabulaire explicatif de la psychologie. Un physicien observant deux humains jouant aux échecs analysera les mouvements en termes d'inertie, d'énergie potentiel (soulever un cavalier), de levier (l'articulation du coude), etc. Le psychologue cherchera à détecter les fluctuations dans les niveaux de concentration et de motivation, il cherchera à décrire les processus de raisonnement et de mémoire qui mènent à une prise de décision (échanger les reines).

Chaque science, idéalement, propose un paradigme de description et d'explication particulièrement adapté à une classe de problèmes. Une science ne traitera pas de phénomènes qui ne tombent pas dans son champ de perception. La richesse propre aux efforts de recherche pluridisciplinaires provient du fait que chaque spécialiste propose un point de vue complémentaire de celui de ses collègues. L'intersection de deux domaines scientifiques éloignés a toujours été une source puissante de progrès et d'idées nouvelles (informatique et psychologie, par exemple).

Un paradigme induit un état d'esprit, un ensemble sélectif de pré-dispositions qui conditionnent notre perception. Nos concepts conditionnent nos percepts, et ce qu'on perçoit modifie notre univers conceptuel. Les concepts selon lesquels on interprète un phénomène déterminent en grande partie les principes structurants qu'on cherchera à identifier : ils dirigent notre attention vers certains attributs et la détournent d'autres attributs. (Il est remarquable, par exemple, qu'avant la découverte des chromosomes, aucun dessin de noyau de cellules n'en comportait alors qu'ils étaient tout à fait visibles).

Il devient clair qu'un paradigme, à lui seul, ne permet qu'une vision limitée de la "réalité". Se restreindre à peu de paradigmes explicatifs est néfaste. Le fanatisme idéologique serait la forme extrême d'une vision-tunnel paradigmatique...

2. PARADIGMES ET PROGRES INTELLECTUEL

Le progrès scientifique semble suivre un schéma répétitif : un paradigme ou vision des choses généralement acceptée par un corps scientifique (eg. la physique newtonienne) se bute à des observations qu'elle ne peut expliquer (eg. le comportement énergétique des corps noirs). Le corps scientifique tente d'ignorer ces observations. Celles-ci se représentent de façon indéniable. Un malaise profond apparaît. Quelques chercheurs (le plus souvent jeunes) proposent une vision radicalement nouvelle qui explique aussi les phénomènes dérangeants. Il y a un mouvement très vif de résistance parmi la "vieille garde" mais le nouveau paradigme finit par prendre le dessus, jusqu'à ce qu'il soit confronté à des phénomènes qu'il ne peut expliquer, et ainsi de suite.

Selon la vision de Jean Piaget, le développement intellectuel chez l'enfant, avec ses processus d'assimilation et d'accommodation, n'est pas sans rapport avec le scénario précédent. Chez l'enfant, les interactions avec l'environnement alimentent d'abord une certaine compréhension des choses. Progressivement, les informations recueillies ne peuvent plus être assimilées par les structures mentales de l'enfant de façon harmonieuse. Ceci provoque un déséquilibre. Ce déséquilibre se résoud par une rupture qui est l'ascension à un niveau supérieur de compréhension.

De façon moins fondamentale, nous effectuons ce type de progrès dans nos apprentissages quotidiens surtout lorsque nous sommes confrontés à des concepts radicalement nouveaux (la récursion, par exemple).

Ce passage à un point de vue radicalement nouveau est aussi un élément essentiel lors d'activités de résolution de problèmes. Cette fois, il ne s'agit plus de passer à un niveau supérieur de compréhension mais plutôt d'accéder à plusieurs points de vue complémentaires afin de trouver celui qui sera le plus productif.

La capacité de passer d'un paradigme de représentation et d'explication à un autre apparaît donc comme centrale à toute évolution intellectuelle et joue un rôle important lors d'activités de résolution de problèmes.

3. ACTIVITES DE PROGRAMMATION ET RESOLUTION DE PROBLEMES

Un langage de programmation permet de décrire des comportements à un ordinateur. Tout comme il existe plusieurs sciences, chacune proposant sa vision des

choses, il existe plusieurs paradigmes de programmation, chacun supportant une façon de décrire des comportements.

Une famille de langages de programmation est l'ensemble des langages qui supportent un même paradigme de représentation. Par exemple dans la famille des langages supportant le paradigme fonctionnel, on retrouve LOGO, LISP, APL, etc.

Chaque paradigme de programmation privilégie un ensemble particulier de stratégies d'analyse et de descriptions. Chacun impose une approche, un point de vue particulier sur tout problème. Certains types de problèmes se traitent plus facilement selon un certain paradigme. Par exemple, les problèmes de simulation gagnent à être abordés d'un point de vue orienté-objet.

Exposer un individu à un seul paradigme de programmation, c'est l'exposer à un sous-ensemble restreint de stratégies de programmation. Cela provoque deux déficiences :

1 - le-dit individu n'est équipé pour affronter efficacement que certains types de problèmes informatiques.
2 - plus le passage à un autre paradigme tarde, plus ce passage devient difficile.

Examinons brièvement quelques-uns des plus importants paradigmes de programmation dans ce qu'ils sont et selon les stratégies de résolution de problèmes qu'ils supportent.

4. DESCRIPTION DE QUELQUES PARADIGMES DE PROGRAMMATION

Paradigme VON NEUMANN

Les concepts et opérations supportées par ce paradigme sont ceux qui décrivent l'architecture et le fonctionnement des ordinateurs traditionnels. Une banque de mémoire divisée en cellules individuellement adressables contient de façon disjointe des instructions et des données. Une instruction à la fois est exécutée par un processeur unique. Les données sont examinées et modifiées une à la fois, de façon séquentielle. Certaines instructions ont pour effet de déterminer quelle sera la prochaine instruction à exécuter (sauts conditionnels, etc).

Le langage binaire et l'assembleur supportent ce paradigme. Il en est ainsi de la plupart des langages qui sont des descendants de Fortran (Basic, Pascal, C, etc), quoique de façon plus abstraite.

Les restrictions qu'impose ce paradigme ont leur origine uniquement dans les contraintes émanant de l'architecture de la machine. Le programmeur doit percevoir tout

problème en termes de séquences d'opérations modifiant une à la fois des données de dimensions fixes.

Le seul avantage qu'offre ce paradigme est qu'il permet de concevoir des programmes très efficaces. Il est autrement très peu "naturel" et impose des contraintes qui sont à priori plus celles de la machine que celles du problème à résoudre.

Le paradigme fonctionnel

Ce paradigme prend son origine dans le langage mathématique traitant des fonctions. Une fonction, dans son incarnation informatique, accepte des données et produit des données. Une fonction peut être soit "primitive" soit formée par une composition de fonctions. Il n'y a pas de séparation entre données et programmes : une fonction est un objet de "première classe" sur lequel d'autres fonctions peuvent opérer. Ce paradigme ne contient pas la notion de variable : un programme écrit dans un langage fonctionnel pur ne produit jamais d"effets secondaires. Les langages LISP, LOGO et APL sont d'abord et avant tout des langages fonctionnels. Pascal et C permettent l'écriture de fonctions.

Ce paradigme propose un point de vue "transformationnel". Tout comportement doit être perçu comme un enchaînement de transformations sur un état initial et produisant un état final. Le concept de transformation que supporte ce paradigme est général et, contrairement au paradigme Von Neumann, transcende son implantation informatique.

La programmation fonctionnelle offre un excellent modèle de décomposition d'un problème, permettant tant une analyse descendante qu'ascendante. Il n'est pas nécessaire de savoir comment est implantée une fonction, seule la transformation qu'elle effectue doit être retenue. Chaque fonction devenant une sorte de boîte noire, il est aisé d'assembler un grand nombre de fonctions pour constituer des programmes de grande dimension.

Malgré sa généralité, l'approche fonctionnelle n'est pas toujours la plus appropriée. Dans un contexte de bases de données, par exemple, le concepte organisateur serait relationnel plutôt que fonctionnel.

Le paradigme logique

Un programme logique est un ensemble d'axiomes (faits et règles de déduction). Un tel programme est "éxécuté" lorsqu'on pose une requête visant à prouver qu'un énoncé peut être déduit à partir des axiomes. L'objet de base est le prédicat (par exemple : couleur (rose, rouge) qui décrit une relation entre un certain nombre d'individus.

Un programme logique correct est une description d'un problème suffisamment complète pour que la solution puisse en être déduite. Un programme logique est déclaratif ; programmer en logique consiste à décrire le "quoi" et non le "comment". Les habiletés mises en pratique sont des habiletés de description et d'analyse logique.

Prolog est le représentant le plus connu des langages de programmation logique. Contrairement à un langage logique pur, Prolog exige de l'usager que celui-ci comprenne le mécanisme de preuve utilisé (unification, retour en arrière, recherche en profondeur, etc). Néanmoins, un programme Prolog peut se prêter, dans une certaine mesure, à une lecture déclarative.

Une grande simplicité syntaxique, la puissance du principe d'unification et le très petit nombre de primitives que doit connaître un usager font que Prolog s'aborde très facilement. Il permet d'accéder très rapidement à des exercices de traitement de l'information autour de bases de données relationnelles.

Il est comparativement très difficile d'apprendre à utiliser Prolog pour traiter des structures de données. Bien qu'un très petit nombre de schémas de solutions récursifs suffisent à traiter un très grand nombre de problèmes, ces schémas doivent être assimilés au niveau formel. La difficulté n'est pas causée par une déficience de la part Prolog mais s'explique du fait que Prolog exige une maîtrise formelle. Cette difficulté ne doit pas rebuter les enseignants d'informatique, au contraire. Nous devons apprendre à utiliser Prolog comme un outil puissant favorisant l'accès à un niveau supérieur de pensée analytique.

Le paradigme orienté-objet

Bien qu'il soit parfois "naturel" de percevoir un problème en terme de relations entre des individus, notre attention semble très souvent se porter d'abord sur les objets peuplant un problème et ensuite sur leurs interactions. Le paradigme orienté-objet utilise l'objet comme principe unificateur.

Un objet a un état qui est l'ensemble de ses propriétés (ou variables d'instance). Un objet a aussi un comportement : un ensemble de réactions (ou méthodes) qui peuvent être explicitées par d'autres objets via l'envoi de messages.

Un objet préserve son intégrité en ne permettant à aucun autre objet de le modifier directement. Un objet peut seul se modifier, à la demande d'un autre possiblement. Les attributs communs d'un type d'objet (propriétés et réactions) sont décrits dans une classe et sont hérités par tous les membres de cette classe. Si cette classe a une ou plusieurs surclasses, ses membres héritent aussi des surclasses.

Programmer avec un langage orienté-objet, tel Smalltalk, amène le programmeur à identifier les "acteurs" qui composent un problème, puis à déterminer ce qu'est et ce que doit savoir chaque acteur. En regroupant les aspects communs puis en les spécialisant, le programmeur établit une hiérarchie de classes. Puis il cherche à décrire quels échanges de messages entre les acteurs produiront les comportements voulus.

Ce mode de programmation est un levier intellectuel très puissant. Il offre, via l'objet, un excellent outil de modularité. En utilisant une hiérarchie de classe, il peut décrire un comportement au niveau de généralité le plus approprié, avec l'économie de

programme que cela procure. Les programmes orientés-objets sont éminemment réutilisables et modifiables. En fait les environnements de programmation orientée-objet offrent un vaste choix de classes pré-définies. Programmer dans de tels environnements revient à trouver ce dont on a besoin ou à spécialiser ce qui s'en rapproche le plus.

Bien que les concepts de base propres à ce paradigme soient intuitifs, acquérir un savoir-faire en programmation orientée-objet est étonnamment difficile. Il y a plusieurs facteurs en cause :

1 - apprendre à se retrouver dans un monde peuplé de quelques centaines de classes prend du temps.

2 - pour comprendre complètement ce qu'est et sait faire un objet, il faut remonter la filière de son héritage.

3 - écrire un programme signifie augmenter un environnement complexe ; cela exige une compréhension minimale des interactions entre des objets de classes pré-définies.

Décrire les propriétés et savoir-faire d'un objet, généraliser ses attributs lors de la création de surclasses, etc... sont des habiletés qui dépassent le contexte de la programmation. Qu'elles soient perçues comme problématiques lors d'activités de programmation a peu à voir avec le langage lui-même : ce sont là des habiletés qui sont en soi difficiles à maîtriser.

5. COMMENT S'EXPOSER A PLUS D'UN PARADIGME DE PROGRAMMATION

On peut s'exposer à plus d'un paradigme de programmation de deux façons :

1 - en utilisant plus d'un langage appartenant à des familles différentes (par exemple, LOGO et Prolog ou Pascal et Smalltalk).

2 - en utilisant des environnements à paradigme multiples.

Les environnements à paradigmes multiples existent sous plusieurs formes :

- un langage auquel on ajoute une ou plusieurs extensions paradigmatiques.

  Lisplog = Prolog en Lisp,

  Prolog/V = Prolog en **Smalltalk/V** de Digitalk,

  SOAP = Objets en **SD-PROLOG** de Systems Designers,

  KEE d'Intellicorp = Lisp + Objets + règles + démons +
  mondes hypothétiques + ...

- un environnement qui intègre plusieurs langages appartenant à des familles différentes.

  POPLOG de Systems Designers = Lisp + Prolog + Pop-11 +
  Flavours + ML

S'exposer à plus d'un paradigme ne se fait pas sans difficulté. S'initier à un paradigme de programmation très différent de celui qu'on maîtrise demande qu'on redevienne un novice. L'effort intellectuel qu'exige l'acquisition d'un nouveau mode de conception et de représentation est considérable et requiert une longue période de maturation. Les habiletés et techniques acquises dans la pratique d'un autre paradigme font interférence ; il est difficile de penser "Prolog" face au problème de l'inversion d'une liste lorsqu'on peut résoudre ce problème en Pascal de façon quasi-automatique.

Cet effort de transition, bien que considérable, tend à provoquer une réflexion de niveau épistémique. Les anciennes habitudes de résolution de problèmes doivent être reconsidérées, et pour cela il faut être en mesure de se les représenter afin de les évaluer objectivement. Tout comme il est difficile de réfléchir sur le langage si on ne parle qu'une seule langue, il est difficile de réfléchir à propos de stratégies de modélisation informatique si on ne connaît qu'un seul paradigme de programmation.

Utiliser plus d'un paradigme de programmation est libérateur et éminemment riche en situations d'apprentissages. Une didactique de l'informatique se doit de considérer le concept de paradigme de programmation comme un des éléments centraux de son discours.