Réflexion sur les langages de la quatrième génération

Réflexion sur les langages de la quatrième génération

par Jean-Paul Racine

Introduction: syntaxe et sémantique

L'idée de communiquer avec l'ordinateur à l'aide de langages évolués n'apparaît qu'au cours du milieu des années cinquante.

A l'initiative de John Backus, qui prend conscience du gaspillage de travail nécessaire à la mise au point des programmes en langage-machine, un groupe de chercheurs d'IBM décide de mettre à disposition des programmeurs un instrument plus ou moins performant de formulation symbolique: FORTRAN (pour FORMULA TRANSLATOR) était né.

A partir de cette époque, le langage-machine qui est propre à chaque ordinateur, difficile à maîtriser et éloigné de notre mode de pensée, disparaît insensiblement au profit des premiers vrais langages de programmation.

Un langage informatique devient un ensemble de règles permettant de réaliser un texte appelé programme.

L'approche utilisée par les scientifiques pour créer ces langages remonte aux travaux de Chomsky sur les grammaires abstraites et structuralistes, qui font passer au second rang des notions telles que les compléments directs ou indirects. Il en résulte deux caractéristiques typiques dans la construction (l'analyse) de la phrase:

a) *la récursivité*

exemple: (i) phrase

(ii) (groupe sujet) (groupe verbal)

(iii) (pronom) (verbe) (adverbe) (complément)

tu chantes bien Mozart

b) *l'indépendance entre validité syntaxique et signification*

exemple: il dévore terriblement les métaux

L'approche structuraliste n'est pourtant pas aussi radicalement neuve que l'on croit, car elle est implicite dans l'œuvre de Saussure et dans la grammaire de Port-Royal.

Et, comme le langage naturel, la programmation, qui est le moyen de dialoguer avec l'ordinateur, procède aussi en partie par essais et erreurs.

Mais contrairement à l'enfant qui manipule des structures syntaxiques fondamentales et construit ses premières phrases par opérations récursives aléatoires, le programmeur est capable d'imaginer les réactions de la machine et de choisir «les mots pour le dire» en fonction du but à atteindre.

Le modèle de Chomsky est de plus strictement séquentiel, comme le langage naturel, et permet l'analyse des langues européennes en recherchant leur conformité respective à un ensemble de diagrammes syntaxiques qu'on appelle grammaire.

Outre des règles de syntaxe, la programmation, comme le français, impose des règles de sémantique et des règles de style.

Si la phrase «LE JOUR? LE PLUS, LONG» est un texte admissible du point de vue du diagramme syntaxique, il est évident qu'elle ne procurerait pas le certificat d'études à son auteur.

Langages séquentiels: langages de troisième génération

La genèse de ces langages informatiques qu'on dit de la troisième génération est contemporaine de la diffusion de l'informatique dans les secteurs d'activités de la recherche, de la gestion et de l'ingéniérie. Ce sont Fortran en 1954, Algol en 1958, Lisp en 1958, Cobol en 1959. L'émergence de la micro-informatique, à la fin des années septante, propage très rapidement, par le biais des écoles, un langage presque oublié, le Basic, inventé en 1964 au Dartmouth College par deux professeurs, Khemeny et Kurtz, à la demande de General Electric. Ce langage sera suivi par la diffusion rapide des langages structurés tels Pascal, inventé par Niklaus Wirth en 1968 à l'Ecole polytechnique fédérale de Zurich, Logo, Modula-2, Forth, Ada et autres C.

A partir de cet instant, c'est l'ordinateur qui assure lui-même, au travers du langage, la gestion de son espace mémoire et qui prend en charge les divers mécanismes d'adressage facilitant la tâche du programmeur, mais augmentant d'autant la complexité de la machine.

Tous ces langages sont séquentiels et décrivent une succession d'actions plus ou moins indépendantes de la machine: lire, écrire, transférer un fichier, calculer une racine carrée, etc.

Les «phrases» ressemblent à celles des langages naturels, l'anglais généralement, parce que la création des langages s'est accomplie parallèlement au développement technologique des microprocesseurs venus des USA.

Dans ces langages, chaque instruction génère, à l'issue d'une phase d'«interprétation» ou de «compilation», une suite d'instructions en langage-machine. Cette traduction d'un langage de haut niveau (programme source) en un code typique pour chaque machine (programme objet) directement exécutable est le fait du système d'exploitation, lequel possède tous les outils d'automatisation et de mise en œuvre des langages classiques évolués.

Dans ce foisonnement de langages de programmation (plus de deux mille sont recensés), peut-on fixer des critères de choix ou des qualités souhaitables minimales? La difficulté de choisir vient de ce que certaines caractéristiques d'un langage apparaissent comme des qualités d'un certain point de vue et des défauts appréciées sous un autre angle. Les avantages et les inconvénients d'un langage ne peuvent être évalués qu'en fonction des applications.

Si la plupart des langages classiques sont assez équivalents, c'est qu'ils subissent une adaptation constante aux progrès et aux nécessités de la convivialité.

La micro-informatique a démontré que la facilité d'apprentissage d'un langage de programmation est une des qualités qui permettent de vaincre certaines barrières psychologiques empêchant leur diffusion.

La facilité d'écriture conduit tout naturellement à la création des structures de blocs et à la modularité des programmes. Elle permet une conception et une réalisation hiérarchisées des programmes et une mise en page avec indentations qui mettent en évidence la logique du programme.

Enfin, il est important de connaître les types et les structures de données que le langage peut définir et manipuler. Ils déterminent évidemment les niveaux d'applications abordables par le langage.

La concision d'un langage informatique peut être une qualité, mais aussi un défaut. Si un langage trop verbeux devient vite fastidieux

(p.ex. Cobol), il ne faut pas que sa concision entraîne une mauvaise lisibilité (p.ex. APL). Les performances du langage informatique ne sont paradoxalement pas un critère de choix, alors que leur portabilité est de plus en plus un indice de la qualité d'un langage.

La beauté du flou: physique qualitative

Si les scientifiques ont, depuis longtemps déjà, défini des algorithmes pour résoudre des classes entières de phénomènes susceptibles d'être représentés numériquement, nos connaissances ayant trait aux questions qualitatives sont beaucoup plus réduites. Les langages procéduraux, passés en revue précédemment, lesquels indiquent toutes les opérations effectuées pas à pas par la machine en considérant comme secondaire la nature et le sens de ces actes, sont mal adaptés à la manipulation des listes et des concepts déclaratifs.

Lorsque nous nous exprimons, c'est souvent pour indiquer le résultat d'une observation, pour poser une question ou y répondre, pour faire une description, etc. Nous détaillons rarement ces processus en opérations élémentaires, précisant à chaque fois le traitement adéquat pour obtenir le résultat.

La plupart du temps, les langages naturels sont bien non-procéduraux et c'est dans la catégorie des langages déclaratifs que se situent les langages de la quatrième génération. Au lieu de décrire un traitement, le programme décrit les propriétés des informations à manipuler et celles des résultats attendus.

A partir de cette description, l'ordinateur va faire une exploration systématique de tout l'espace des solutions pour ne retenir que celles qui sont compatibles avec les propriétés. Ces propriétés sont énoncées sous la forme d'implications logiques entre des objets représentant des relations.

Ce formalisme permet de représenter aussi bien les données que les propriétés, et les mécanismes de recherche des objets satisfaisant les propriétés sont complètement transparents à l'utilisateur et entièrement pris en charge par l'interpréteur du langage.

La première apparition en informatique de cette approche date de l'invention des tableurs: Visicalc, Multiplan, Lotus 1-2-3, etc. Conçus à l'origine pour l'analyse financière, leur structure s'adapte à la description d'une part importante du processus. Un tableur, constitué de lignes et de colonnes, dont l'intersection définit des cases ou cellules, n'indique aucun enchaînement temporel. Le contenu des cellules n'est

pas une suite de pas à exécuter conduisant à la solution du problème, mais une structure statique englobant l'ensemble du processus.

L'utilisation d'un tableur ne fait pas appel à la notion d'algorithme, bien que celle-ci soit sous-jacente dans les programmes de calculs de formules intervenant dans les tableurs. Aussi une telle structure est-elle à la base des langages non-procéduraux qui englobent d'autres types de représentation, notamment les diagrammes en arborescence.

L'informatique de Monsieur «Tout le Monde»

C'est la nécessité de rendre l'utilisation de l'outil informatique accessible aux non-informaticiens qui est véritablement à l'origine des langages de quatrième génération. Ces derniers permettent un dialogue en langage quasi naturel avec les bases de données et optimisent les interactions Homme-Machine. Ils visent donc tous à rendre les utilisateurs du programme indépendants de l'informaticien pour consulter des fichiers, modifier un programme ou formuler et résoudre un nouveau problème.

L'essor que connaissent ces langages depuis les années quatre-vingt est dû au fait que les prix des logiciels dépassent largement ceux du matériel. Il a fallu trouver un moyen de développer des programmes plus rapidement, plus commodément et surtout plus économiquement.

La maintenance du logiciel accroît de plus la dépendance des entreprises par rapport aux développeurs de programmes et entrave de ce fait la mobilité des fonctions décisionnelles.

Le concept fondamental de ces nouveaux langages est né de la constatation que, dans tout programme, un certain nombre de fonctions sont très fréquemment répétées, tels l'enregistrement, la recherche, le tri de données, etc. Ces langages intègrent de plus des macro-commandes permettant d'écrire des applications complètes d'une manière structurée en phases élémentaires hiérarchisables, telles que la description des structures dictionnaires[1], la création des grilles d'écrans, la génération des procédures de traitement, la génération des rapports entre les différents fichiers, la création des menus, des aides et

[1] Une structure dictionnaire est l'utilisation correcte d'un mot dans un contexte donné.

des sécurités. Ils n'autorisent toutefois pas l'économie de la conception fonctionnelle du programme, mais permettent en revanche à l'utilisateur de se concentrer sur cette phase la plus importante, en le dégageant de la rude tâche de la programmation.

L'utilisateur s'attachera donc à bien exprimer le problème à résoudre; l'ordinateur se chargera d'optimiser sa solution. Car un responsable d'entreprise, un chercheur scientifique, un étudiant universitaire ne peuvent pas être à la fois informaticien et autre chose.

Les bases de données

Plus connus sous l'abréviation L4G, les langages de quatrième génération sont généralement associés à des bases de données relationnelles.

Dans la préhistoire de l'informatique (les années soixante à septante), on ne connaît pas la notion de base de données. Toutes les informations sont stockées dans des fichiers (support magnétique) et gérées par des programmes gestionnaires de fichiers interfacés avec des programmes d'application.

Si un programme de gestion utilise les données d'un fichier de stock par exemple, un programme de statistique désirant exploiter les mêmes données devra invariablement réorganiser le fichier STOCK à sa manière, car le rangement physique des informations dépend de l'application qui les exploite. Cela conduit à dupliquer les données dans plusieurs fichiers; cette multiplication rend très vite impraticable une mise à jour cohérente des informations.

Il a bien fallu imaginer des systèmes plus évolués, les SGBD pour les informaticiens (comprendre Système de Gestion de Base de Données). Ceux-ci privilégient une organisation cohérente et logique des informations, indépendante des besoins des diverses applications ultérieures et de l'emplacement physique des données sur le support magnétique par exemple.

Si l'avantage déterminant est l'exclusion pratique des redondances et de ce fait un gain de place, la facilité de mise à jour limite les erreurs qui se propagent par la suite comme des virus dans tous les fichiers.

Au fil des ans, les schémas logiques d'organisation des informations n'ont pas manqué d'évoluer, donnant naissance successivement à trois catégories de bases de données, aujourd'hui encore toutes usitées: les SGBD de type «hiérarchisé», ceux de type «réseau» et ceux de type «relationnel».

Les deux premiers SGBD font appel aux langages de troisième génération pour leur exploitation, c'est-à-dire qu'il est nécessaire d'indiquer au SGBD le chemin qu'il doit suivre pour accéder à une donnée.

Les SGBD relationnels sont en revanche basés sur un L4G non procédural, par exemple SQL (Structured Query Language), qui est un langage universel conçu spécifiquement pour créer, organiser, gérer et interroger des bases de données relationnelles.

Un exemple simple fera comprendre les différences fondamentales entre ces SGBD. Un SGBD hiérarchisé ou réseau est comparable à un chauffeur de taxi sachant parfaitement conduire, se trouvant devant la gare de Cornavin, et auquel il faudra décrire le parcours pour atteindre l'aéroport de Cointrin en désignant sans erreurs la suite des noms des rues qu'il devra parcourir pour atteindre le but. Leur usage nécessite la rédaction de protocoles algorithmiques souvent très longs que seuls les informaticiens chevronnés savent manipuler.

Avec un SGBD relationnel, il suffit de crier au chauffeur «Cointrin» pour qu'il vous y conduise en évitant les rues barrées pour cause de réfection ou les bouchons dus aux heures de midi, quitte à faire un petit détour pour minimiser le temps de parcours.

La manière de procéder d'un SGBD relationnel est pour le moins surprenante et aux antipodes de l'organisation rigide des fichiers.

Le concept en est pourtant simple. La structure logique d'une base de données relationnelle est constituée uniquement de tableaux à deux ou plusieurs entrées appelés tables ou matrices.

Toutes les données s'y retrouvent «à plat». Ainsi les lignes d'une table nommée «TAB-Villes» correspondent chacune à une ville; la deuxième colonne contient des numéros postaux, la troisième contient des nombres d'habitants, etc. Avec une telle organisation, il n'est plus nécessaire d'utiliser une recherche séquentielle pour accéder à un enregistrement; il suffit d'indiquer un index de croisement d'une ligne et d'une colonne pour s'introduire dans la fiche recherchée.

Mais contrairement aux tableurs où les valeurs sont définies par leur position géométrique dans le tableau (ligne/colonne), les L4G ne retiennent dans les tables que les seules variables nécessaires à la «navigation» dans la base de données. Toute modification effectuée sur une variable se répercute automatiquement sur toutes les grandeurs répertoriées.

Par le nombre d'ordres réduits (quelques dizaines) et par l'évidence de leur syntaxe, les L4G émerveillent l'informaticien qui a débuté par l'étude des langages classiques; leur concision s'explique par le carac-

tère non-procédural du langage. Une seule commande peut contenir les opérations les plus complexes à effectuer sur une base de données.

Si l'idée est si simple, pourquoi a-t-il fallu attendre les années quatre-vingt pour la réaliser?

En informatique, la simplicité est souvent trompeuse. Qui dit facilité d'emploi pour l'utilisateur dit aussi haut niveau de complexité pour le logiciel, sophistication et puissance accrues pour le matériel.

Les premiers L4G, tels que ORACLE, INGRES, POWERHOUSE ou RAMIS II, ont été conçus pour de gros ordinateurs; il ne s'est pas écoulé deux ans pour retrouver sur le marché de la micro-informatique des nouveautés équivalentes et traditionnellement réservées aux ordinateurs centraux.

Le représentant de cette nouvelle génération est sans conteste dBase, qui a su associer le premier puissance et simplicité et qui continue d'intégrer les découvertes technologiques récentes comme les générateurs d'applications et le fenêtrage. Les émules de dBase sont fort nombreux et souvent plus performants encore, tels OPEN ACCESS, PARADOX, SUPER DB ou autres SUPERBASE, pour ne citer que les plus importants.

Avec ces L4G, il est devenu possible de traiter des applications de toute nature, faisant, la plupart du temps, l'objet de logiciels spécialisés: dépouillement d'enquêtes, décompte de salaires et paiement multistatuts, facturation, graphisme, calcul de programmes linéaires avec fonctions mathématiques intégrées, mailing, etc.

Un L4G moderne se doit de résoudre les besoins opérationnels et décisionnels d'une entreprise en offrant des solutions totales, intégrées et cohérentes. Son caractère non-procédural doit permettre la résolution d'équations ou de problèmes sans ordre préétabli pour les variables. Ainsi, en application financière par exemple, le langage doit posséder des instructions permettant le calcul d'expressions financières et leur représentation graphique immédiate, les fonctions de consolidation, les générations de rapports, l'analyse des trends par extrapolation, etc.

Un certain nombre de fonctions puissantes facilite le traitement automatique des fichiers, car il s'avère très souvent nécessaire de les lier pour que les informations puissent circuler de l'un à l'autre. Pour éviter, à ce niveau, de recourir à un langage de programmation, plusieurs L4G invitent l'utilisateur à remplir un tableau ligne par ligne, ce que chacun peut mener à bien avec un peu d'attention. Si tant est que le bon sens existe encore, il est ici pleinement sollicité, et c'est faire de la programmation sans le savoir, comme M. Jourdain faisait de la prose.

Les applications personnalisées

Les avantages des L4G sont tels qu'on peut prévoir la disparition de COBOL à très court terme par exemple. Ils simplifient l'écriture des programmes et les rendent bien plus courts. Des applications personnalisées peuvent voir le jour en quelques minutes et sont faciles à relire et à comprendre pour un tiers, lequel peut faire des mises à jour ou assurer la maintenance. La souplesse de ces langages rend possible le passage rapide d'un modèle à un autre et d'en modifier tous les agrégats, donnant accès au programme et aux variables dans des conditions identiques à un traitement de texte. Enfin, ils se caractérisent tous par leur grande interactivité et une extrême convivialité.

Si, lors d'une approche de prototype, ils présentent parfois l'inconvénient d'être moins performants que les programmes écrits en langages classiques, principalement au niveau des temps d'exécution, le talent du programmeur n'est jamais aliéné.

Information, entreprise et décision

A l'heure actuelle, le développement de toute entreprise est étroitement lié à sa capacité d'informer et de se tenir informée. Encore faut-il qu'elle ait les moyens de maîtriser le flux d'informations, d'en organiser le classement, d'en faciliter l'accès et la prise en compte.

C'est à la fin de 1970 que E. Codd, qui est le père des bases de données relationnelles, en définit les principes fondamentaux dans le premier article du livre *A relational model of data for large shared data banks*.

Il faut bien reconnaître que, dans notre vie quotidienne, le rôle des connaissances disponibles et leur utilisation systématique comme base de décision ne cessent de croître en importance. On en est arrivé au point où le savoir est plus important que le raisonnement, et la connaissance est devenue synonyme de pouvoir. Face à la pléthore de renseignements circulant dans le monde grâce à l'essor des télécommunications, leur dissémination et leur complexité s'accompagnent d'une incertitude grandissante. La connaissance globale et les théories objectives faisant défaut, les décisions ne se prennent plus qu'à l'aide d'une connaissance fragmentaire des faits, engendrant souvent des méprises.

Les L4G deviennent le trait d'union vers l'information où la logique est remplacée par des règles empiriques fondées sur l'expérience et le jugement d'experts. La complexité et l'incertitude se gèrent dès lors

par l'expérience et non plus par la théorie. On entre ici de plein pied dans le domaine de l'«Intelligence Artificielle», dont le but est de faire exécuter par des machines les tâches que l'on considère comme intelligentes quand elles sont accomplies par l'homme.

Mais les enjeux politiques, économiques et stratégiques de l'«Intelligence Artificielle» relèvent d'abord d'une réalité technologique, ensuite d'une capacité d'innovation dans un large spectre.

Dès 1982, les Japonais investissent un milliard de dollars dans cette voie et sont suivis, en 1983, par le Pentagone injectant 600 millions de dollars, alors qu'en 1984, 21 entreprises de l'industrie privée américaine leur emboîtent le pas en investissant 65 millions de dollars annuellement sur les dix années à venir.

La Communauté Européenne ne réagit qu'en 1986, en lançant le programme «ESPRIT» (European Strategic Program for Research in Information Technology»).

Une nouvelle aventure scientifique, faite de rêves et de réalités, débute ici et ne peut être abordée dans ce cadre volontairement restreint.

Les L4G de demain permettront de réaliser l'impossible de façon simple, et avec rapidité.

En d'autres termes, les SGBD relationnels gèrent les besoins contradictoires de l'utilisateur, en morcelant de manière toute cartésienne les difficultés d'une application et en proposant une colle magique, telle la lampe d'Aladin, pour réunir les données selon des desiderata sous forme de contraintes multiples permettant d'obtenir des résultats élaborés.

L'informatique de l'an 2000 peut paraître paradoxale à l'observateur de cette décennie. Les ordinateurs se seront rapprochés de l'homme en rapprochant la programmation de la linguistique. Comme le français, les mathématiques et une deuxième langue vivante, l'informatique de l'an 2000 sera une formation indispensable à l'exercice d'une profession tant soit peu intellectuelle et OCCAM[2], devenu vieux, prendra à sa charge la gestion du parallélisme dans le modèle relationnel des bases de données.

Jean-Paul Racine

[2] OCCAM: un des derniers-nés des langages informatiques dont la philosophie est basée sur la gestion des processus et des communications entre processeurs (transputer). Les fonctions de ce langage sont en nombre minime et permettent des exécutions de programme au sein d'un réseau de transputer travaillant en parallèle.

VARIA