CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS

Chaire de Génie Logiciel

Leçon Inaugurale

PROBLÈMES FUTURS DU GÉNIE LOGICIEL

L'informatique et ses limites

Jacques PRINTZ, le 11 Mai 1995

SOMMAIRE

En effet, pour pouvoir examiner avec fruits les principes d'une science, il faut être familiarisé avec ses théories particulières; seul, l'architecte qui connaît à fond, dans tous leurs détails, les diverses destinations d'un bâtiment, sera capable d'en poser sûrement les fondations.

David Hilbert, Les 23 Problèmes: VI, Le traitement mathématique des axiomes de la physique.

The great progress in every science came when, in the study of problems which were modest as compared with ultimate aims, methods were developped which could be extended further and further.

…

The sound procedure is to obtain first utmost precision and mastery in a limited field, and then to proceed to another, some that wider, and so on.

…

The experience of more advanced sciences indicates that impatience merely delays progress, including that of treatment of the «burning» questions.

There is no reason to assume the existence of shortcuts.

John von Neumann, Theory of Games and Economic Behaviour

Le génie logiciel et sa problématique

Périodiquement, mais à un rythme toujours soutenu, l'industrie informatique continue à défrayer la chronique, soit par ses succès: l'ordinateur personnel qui est une vraie révolution, l'espace, les télécommunications, l'imagerie médicale, la réalité virtuelle, etc., soit par ses échecs dont il serait cruel de rappeler quelques exemples récents.

L'amélioration constante, et tout à fait extraordinaire, des technologies informatiques accroît toujours plus la sphère de son influence dans des domaines où l'ordinateur était encore absent, donnant ainsi l'impression d'un progrès ininterrompu, où aucune limite n'est directement perceptible.

L'immensité du succès occulte les échecs, et masque les difficultés, pour des raisons économiques et commerciales évidentes, mais pour combien de temps encore!

Cette leçon inaugurale est l'occasion de jeter un regard lucide sur la nature profonde de la réalité technologique sous-jacente, qui constitue la «matière» de base de l'informatisation, et sur le procédé de fabrication permettant la mise en forme de cette étrange matière.

Retour à l'origine

Le génie logiciel a beaucoup évolué au cours de ces vingt-cinq dernières années. L'une des premières définitions[1], données par D.Tsichritzis, disait: «*The idea behind software engineering is to develop tools, which enable a person of average competence and intelligence to produce good work. Real "hero" programmers are few, although many think they are*». Cette définition

[1] Cf. *Advanced course on software engineering*, Chap 3D, Université de Münich, 1973

est intéressante à plus d'un titre car elle établit une relation entre la compétence de l'ingénieur, la qualité et la nature des outils qui lui seront utiles.

À cette époque, les équipes étaient de taille réduite et l'emphase était mise sur l'individu et sa production personnelle, essentiellement la programmation. Les ressources matérielles: mémoire, temps UC, débit d'entrées-sorties, étaient rares et chères, limitant ipso facto la taille des programmes et des projets.

La révolution de la décennie 80

L'explosion du marché informatique a profondément bouleversé la situation de l'individu producteur, et ceci de deux façons:

Croissance de la population des programmeurs

Une croissance exponentielle du nombre des équipes et des programmeurs, avec disparition de la scène industrielle du programmeur solitaire:

- La cellule productrice est une équipe de 5-7 personnes.
- Apparition de nombreux problèmes de communication entre les équipes, et multiplicité des langages.
- Faible croissance de la productivité individuelle[2] malgré les outils fondamentaux que sont les langages, les bases de données, les réseaux.

Croissance de la taille des projets

La disponibilité de ressources matérielles beaucoup plus abondantes a permis d'envisager la résolution de problèmes beaucoup plus complexes, requérant une masse de programmation beaucoup plus grande, et conséquemment une croissance considérable de la taille des équipes chargées de les développer et de les maintenir, d'où:

- Un considérable problème d'organisation des équipes et de l'architecture du processus de développement permettant d'assurer une productivité correcte. La satisfaction des usagers et la finalité du système à construire s'estompent derrière des préoccupations purement techniques.
- Le génie logiciel devient un enjeu stratégique. C'est le règne des grands projets comme Ada, CAIS, STARS, ESPRIT/PCTE et alii, EAST,… où l'argent a coulé à flot.

Une logique de la communication

La développement d'un grand système produit un flot d'information — messages, transactions, contrats,… — intense entre les individus. La difficulté consiste à éviter que les efforts des uns soient neutralisés par les efforts des autres. Il faut faire en sorte que:

$$1 + 1 \cong 2 \text{ et non pas } 1 + 1 < 1.$$

D'une façon générale, si N individus coopèrent, il faut faire en sorte que la productivité de 1 individu parmi N, reste le plus voisin de 1 possible, mais bien

[2] Cf. ACM SIGSOFT, Vol 8, N° 2 April 83; selon cette étude la productivité moyenne d'un programmeur a été multipliée par 3.6 en 30 ans.

sûr ne sera jamais 1. F.Brooks[3] a expliqué, il y a presque 20 ans, comment cette productivité pouvait être nulle, voire négative.

La mise en place de règles de management facilitant le développement de comportements coopératifs est donc un enjeu majeur du bon fonctionnement des grandes équipes, disons à partir d'une vingtaine d'individus. Toute «optimisation» qui favorise un individu au détriment des autres, ou d'un groupe par rapport à d'autres groupes, est globalement négative sur l'ensemble.

La singularité du génie logiciel

L'histoire des procédés de fabrication nous montre que l'homme, initialement très intégré aux procédés, est progressivement éliminé, au bénéfice de machines et/ou de robots dont il n'assure plus que le contrôle, voire simplement la programmation.

Il semble à peu près certain que l'homme restera indispensable, pendant encore longtemps, à la bonne mise en oeuvre du processus de développement du logiciel.

Pourquoi ce qui a été possible pour la fabrication des circuits, ne l'est certainement pas pour des programmes?

Vis-à-vis du sens qu'ils portent, ces deux artefacts ont un statut très différent. Les circuits sont des objets du monde physique auquel ils restent totalement soumis[4], même si pour les élaborer il faut maîtriser des théories physiques très sophistiquées. L'ingénieur n'a fait qu'utiliser les lois de la nature à son profit. Le programme, quant à lui, est le fruit de la pensée de son créateur: c'est du sens à l'état brut. L'expression de cette pensée, à travers les différents langages de l'informatique, n'a de vérité que par rapport à la réalité dont elle constitue un modèle. Vis-à-vis de cette réalité, le langage informatique va donc fonctionner comme un métalangage permettant la construction d'entités — termes primitifs, axiomes — dont la composition, selon la méthode axiomatique, constituera précisément une modélisation de la réalité[5]. Le programmeur est donc soumis à tous les aléas de l'activité langagière: ambiguïté, paradoxe, inconsistance, incomplétude, indécidabilité, indétermin-isme,…

Observation première

L'homme est indispensable au procédé de fabrication du logiciel.

Conséquence

Le résultat du processus, et le processus lui même, vont contenir un certain nombre de défauts dont l'origine est l'erreur humaine.

La place de l'informatique dans la société

---

[3] F.P.Brooks *The mythical man-month*, Addison-Wesley

[4] *Les frontières physiques de la microélectronique* La Recherche, N° 203, Oct 88.

[5] Cf. le texte lumineux de A.Tarski *Vérité et langage formalisé* dans *Logique, Sémantique, Métamathématique*, Armand Colin.

Au cours des vingt-cinq années écoulées, le rôle et la place des systèmes informatiques dans notre vie sociale se sont profondément transformés. Il y a un vrai phénomène de percolation entre les systèmes existants, et des systèmes encore en gestation qui vont directement pénétrer dans la sphère de la vie privée, dont le Minitel n'est qu'une pâle préfiguration.

Les systèmes informatiques sont déjà présents dans notre environnement économique, ils arrivent à grands pas dans notre environnement humain privé. Du même coup, ils en subissent toutes les contraintes: ils doivent s'adapter, ou mourir, au même rythme que nous, se développer et croître, au même rythme que nous, tout ceci sans dégradation du contrat de service qui définit leur utilité.

Les systèmes informatiques doivent être protégés des aléas de l'environnement. Leur comportement doit rester globalement déterministe. Comme les systèmes biologiques, avec lesquels ils développent une certaine analogie, les systèmes informatiques doivent maintenir un certain nombre de constantes internes: ils doivent être dotés d'invariants dynamiques, analogues aux constantes homéostatiques des systèmes biologiques, comme par exemple: l'intégrité, la sûreté de fonctionnement, la sécurité, la reconfiguration, etc.

Observation seconde

Les systèmes informatiques, dans la mesure où ils cohabitent avec des humains, ou fonctionnent dans des environnements hostiles, doivent être dotés de mécanismes compensateurs et/ou réparateurs permettant de répondre aux aléas de l'environnement externe et interne, imprévisibles par définition.

Conséquence

La cohabitation étroite avec l'homme exige un niveau de fiabilité très supérieur à celui du système, ou de la pratique humaine, qu'il remplace. Faute de quoi le système informatique est incompatible avec le rôle social qui lui est dévolu.

La mécanique d'exécution et les nouvelles architectures

Les programmes qui constituent la partie réactive et opératoire du système informatique, et qui lui donnent son sens, sont exécutés grâce à des composants matériels dont l'évolution a été tout à fait extraordinaire, et presque inverse de celle observée pour le logiciel. Le génie «matériel» a relevé, et complètement résolu, un certain nombre de défis qui paraissaient vraiment impossibles à tenir, et à l'extrême limite de nos savoir-faire:

- Le défi de la miniaturisation.
- Le défi de la fiabilité — grâce à la théorie du codage et des codes correcteurs d'erreurs, omniprésents dans le matériel — qui permet de corriger et compenser les aléas[6], nombreux, auxquels sont soumis les circuits.

[6] Entre autres: les particules α, les bruits thermiques et electromagnétiques, les phénomènes de dislocations des cristaux, etc.

Sans doute, cette évolution a été facilitée par l'extrême stabilité d'interfaces aussi fondamentales que le jeu des instructions de la machine[7], ce qui permet de se concentrer sur l'amélioration du processus de développement, et d'inventer des mécanismes accélérateurs comme les *pipe-lines* ou les caches. C'est certainement une voie à suivre pour le logiciel, comme par exemple stabiliser, une bonne fois pour toute, les langages de programmation.

Les supercalculateurs, les ordinateurs personnels, les réseaux locaux, les RAID[8], les ordinateurs massivement parallèles comme la Connection-Machine ,… témoignent quotidiennement de l'extraordinaire capacité de réplication et d'interconnexion des composants matériels.

Mais l'objet matériel ainsi obtenu est-il pour autant programmable, donc exploitable et utile?

Observation troisième

L'immense succès du matériel permet l'interconnexion de ressources de traitements à une échelle encore jamais vue. Il y a, de fait, abondance de ressources, mais elles se présentent à nous de façon fragmentée, ce qui repose le problème de la communication, précédemment évoqué dans un autre contexte.

Conséquence

Comment mettre ce potentiel à disposition des usagers? Comment adapter nos langages et nos méthodes? Comment cacher l'énorme complexité sous-jacente sans nuire à la portabilité des applications?

Les problèmes futurs

Erreurs humaines, sûreté de fonctionnement, exploitation des nouvelles architectures, insertion de l'informatique dans la sphère privée: voilà quelques défis auxquels nous sommes d'ores et déjà confrontés.

Saurons nous les résoudre, comme nos collègues du matériel ont résolu les leurs? A défaut, — et il faut éviter l'analogie fallacieuse bien que très tentante avec le matériel, car le logiciel a des différences profondes, essentielles et subtiles avec le matériel —, saurons-nous tracer la frontière à ne pas dépasser, au delà de laquelle le système devient socialement inacceptable?

Résoudre des problèmes est une méthode particulièrement féconde pour faire progresser les sciences fondamentales, tout autant que les sciences appliquées comme les sciences de l'ingénieur. J'ai la conviction que de grands progrès peuvent être réalisés, pour autant que l'on sache ancrer nos analyses dans la réalité industrielle, observer cette réalité, tirer en toute lucidité les leçons de nos succès et de nos échecs, être conscients de nos limites.

---

[7] Cf. *Case study: IBM's system/360-370 architecture*, Communications of the ACM, Vol 30, N° 4, April 87, dont le jeu d'instructions est un record absolu de longévité.

[8] **R**edundant **A**rray of **I**nexpensive **D**isk

Je formulerais, pour ma part quatre problèmes auxquels j'entends m'attacher tout particulièrement dans les années à venir.

Problème N° 1

Comment éviter, détecter, compenser les erreurs humaines?

Problème N° 2

Comment détecter et compenser les défaillances des systèmes informatiques, quelle que soit l'origine de la défaillance, interne ou externe, fonctionnelle ou technique?

Problème N° 3

Peut-on raisonnablement tirer partie, et comment, des nouvelles architectures matérielles, en particulier à la lueur des problèmes 1 et 2?

Problème N° 4

Quelles sont les implications sociales des modes de construction des systèmes? Sont-ils de vraies boîtes noires?

L'erreur humaine

Le constat

L'homme étant par nature faillible, l'erreur humaine ne doit pas être considérée systématiquement comme un mal honteux mais comme une éventualité à laquelle peut être attachée une certaine probabilité. La vraie difficulté est de mettre en place une méthodologie capable de détecter les dysfonctionnements humains et d'en limiter les effets.

L'examen des défaillances humaines qui peuvent plus ou moins entacher les actes et décisions survenant tout au long du cycle de développement du logiciel conduit à retenir un certain nombre de types d'erreurs:

— erreur de perception,

— erreur de codage/décodage,

— non-respect d'une procédure ou d'une règle,

— erreur de communication homme-homme, de traduction lors d'un changement de code,

— non-prise de décision en temps voulu,

— action non-adaptée au contexte,

— erreur de raisonnement, défaut de généralité,

— erreur de représentation et abstraction mal construite

— compréhension et représentation erronées de phénomènes dynamiques et/ou combinatoires.

Ce découpage un peu théorique, mais néanmoins exhaustif, permet de se rendre compte que l'activité liée au développement du logiciel est largement tributaire des mécanismes psycho-cognitifs sous-jacents à cette typologie. D'où un taux d'erreurs par programmeur très au dessus de la moyenne de ce que l'on peut constater dans d'autres disciplines moins exigeantes. Tout ceci ne va pas sans poser de très graves problèmes, tant au niveau des individus qui peuvent se

sentir culpabilisés à l'extrême, que des organisations qui vont avoir tendance à masquer ou nier le phénomène.

La NASA, une des rares organisations à avoir publié des statistiques, fait état de taux d'erreurs découvertes après livraison des logiciels de la navette spatiale[9]:

— logiciel embarqué: 0.11/KLS pour 500 KLS, soit 55 erreurs sur la partie la plus critique,

— logiciel sol: 0.40/KLS pour 1400 KLS, soit 560 erreurs.

Sur des systèmes d'exploitation[10], diffusés à de très grands nombres d'exemplaires, le flux des erreurs découvertes en exploitation est de plusieurs milliers d'erreurs par an.

Si l'on essaye d'estimer le nombre d'erreurs probables lorsque les programmes sont correctement compilés, il faudrait probablement multiplier ces chiffres par cent, ce qui conduirait à une erreur toutes les 50-100 lignes. Dans une activité plus littéraire, ce ne serait pas une trop mauvaise performance!

Les remèdes

Le développement des arts et métiers, des techniques, puis des sciences ont été, et sont encore une lutte permanente contre l'erreur. De cette histoire, on peut tirer un certain nombre de leçons.

Le geste, prolongé et relayé par l'outil, répété inlassablement des milliers de fois devient un instrument de précision, capable d'accomplir des tâches d'une extrême complexité. C'est là la grandeur de l'apprentissage. Maîtrisant parfaitement son geste, le compagnon peut alors entreprendre son chef d'oeuvre.

Les technologies modernes, quant à elles, sont une preuve existentielle de notre capacité à résoudre le problème des défaillances, pour autant qu'on y mette l'énergie et l'attention nécessaire. Dans le domaine du traitement de l'information, les résultats ont été particulièrement spectaculaires. On peut citer:

— la transmission de l'information,

— le stockage de l'information,

qui sont deux domaines où le temps moyen de bon fonctionnement[11] des organes a été amélioré dans des proportions allant de quelques heures à l'origine, à plusieurs années aujourd'hui, soit un gain de plus de 10.000.

---

[9] Ces logiciels sont, à bon droit, considérés par la communauté scientifique comme les mieux testés du monde. 1 erreur par KLS est généralement considéré comme excellent.

[10] Ce sont, de très loin, les plus grands logiciels actuellement en service; leur taille oscille entre 5-10 millions de lignes source, soit au minimum 200 volumes de 500 pages, rien que pour le texte source!

[11] C'est le MTTF, *Mean Time To Failure*.

La méthode utilisée pour résoudre les nombreux problèmes que posait la fiabilité de ces technologies, sans lesquelles l'ordinateur n'existerait pas, a toujours été la même:

— compréhension en profondeur des phénomènes physiques sous- jacents,

— organisation «intelligente» de la redondance des mécanismes critiques.

Les deux technologies précitées font un large usage de la théorie mathématique de la communication[12], et des deux théorèmes de Shannon que tout informaticien d'un certain niveau devrait connaître et méditer en permanence. En substance, le deuxième théorème tire partie du fait que l'occurrence simultanée de deux erreurs aléatoires affectant un même mécanisme est un événement beaucoup plus rare que la simple panne. En terme opératoire, la probabilité de panne d'un mécanisme fonctionnel ET d'un «observateur» de ce mécanisme conçu indépendamment, est beaucoup plus faible que celle du mécanisme fonctionnel seul.

L'art de l'ingénieur va donc consister à placer des observateurs aux «bons» endroits, et en quantité «suffisante» pour donner à l'ensemble un niveau de fiabilité compatible avec la mission du système. Pour ce faire, il doit connaître exhaustivement, et maîtriser parfaitement, l'architecture du système qu'il construit.

La science, quant à elle, nous fournit trois types de méthodes qui devraient être les piliers de toute formation d'ingénieurs.

Premier pilier: la méthode axiomatique

La rigueur de raisonnement est l'une des armes traditionnelles de l'ingénieur. Faire des déductions fondées sur des faits, généraliser des éléments parcellaires en un tout cohérent, poser correctement un problème,… autant d'éléments de logique dont le point d'aboutissement est la méthode axiomatique.

Point n'est besoin de mathématiques sophistiquées pour la mettre en pratique: les mathématiques élémentaires suffisent.

Second pilier: la méthode expérimentale

On peut la résumer en quatre verbes: observer[13], mesurer, modéliser, tester. Dans l'univers incertain et sémantiquement ouvert où nous sommes, le recours à l'expérience est souvent l'unique façon de vérifier si ce que l'on dit est vrai.

Là encore, l'étude de quelques phénomènes élémentaires suffit pour s'en faire une idée très précise. Mais il faut mener la démarche de A à Z, et ne pas se contenter d'énoncer les lois qui n'en sont que le résultat.

Troisième pilier: la méthodologie des systèmes

À la différence du scientifique qui peut parfois s'isoler dans sa discipline, l'ingénieur est toujours confronté à une pluralité de points de vues, tous aussi valides les uns que les autres, mais parmi lesquels il devra faire des choix et

---

[12] C.Shannon, W.Weaver *The mathematical theory of communication.*

[13] On oublie trop souvent que la méthode scientifique débute toujours par l'observation des phénomènes. Maxwell est inconcevable sans Faraday qu'il admirait!

trouver des compromis. Une organisation mal adaptée à la finalité du produit à construire peut être fatale au développement, quels que soient les mérites techniques des individus qui la composent.

De plus, toute action humaine se déroule dans le temps. Il est donc très souhaitable d'avoir de bonnes notions sur la dynamique des systèmes[14], surtout si les interactions entre les éléments du système déclenchent des rétroactions, ce qui est le cas général des systèmes informatiques, et tout particulièrement du logiciel.

L'action

Une bonne façon de diminuer le nombre d'erreurs est d'abord de les éviter. Mais comme il en restera toujours, il faut donc les détecter, puis compenser les dégâts qu'elles ont pu occasionner.

Éviter les erreurs

La meilleur façon d'éviter les erreurs est d'être parfaitement renseigné sur ce qui marche et à fait ses preuves. Il faut connaître le maximum d'algorithmes[15] qui sont les briques de bases de l'activité de programmation.

Lorsque l'on est obligé de créer des algorithmes, l'aptitude au raisonnement est un prérequis absolu.

Enfin, si l'on a le choix entre deux programmeurs, il vaut mieux confier le travail a celui qui commet le moins d'erreurs, si tant est qu'on le sache!

Détecter les erreurs

C'est le rôle fondamental des méthodologies logiciel. Une méthodologie qui n'a pas comme objectif premier de faciliter la détection des erreurs ne sert à rien, et est même tout à fait nuisible.

On peut classer les textes qui sont le produit du processus de développement en deux catégories: ceux qui sont exécutables sur un ordinateur, et ceux qui ne le sont pas.

Sur les seconds on peut appliquer la méthode, qui est celle généralement utilisée par les mathématiciens, consistant à faire relire sa production par des pairs. Elle a été systématisée sous la forme d'inspections, de revues et de certifications.

Sur les premiers, on peut se livrer à deux types d'activités complémentaires:

— Élaborer des tests qui doivent alors être conçus comme des protocoles expérimentaux parfaitement reproductibles.

— Effectuer, sur le texte lui même, des vérifications permettant de s'assurer mécaniquement, par calcul, que le programme possède des propriétés intéressantes.

---

[14] Cf. La collection Recherches Interdisciplinaires, Maloine-Douin Ed, qui contient quelques ouvrages interressants et très accessibles, à coté des textes fondateurs de N.Wiener, R.Ashby, Von Neumann.

[15] D.Knuth *Art of computer programming* reste une référence incontournable.

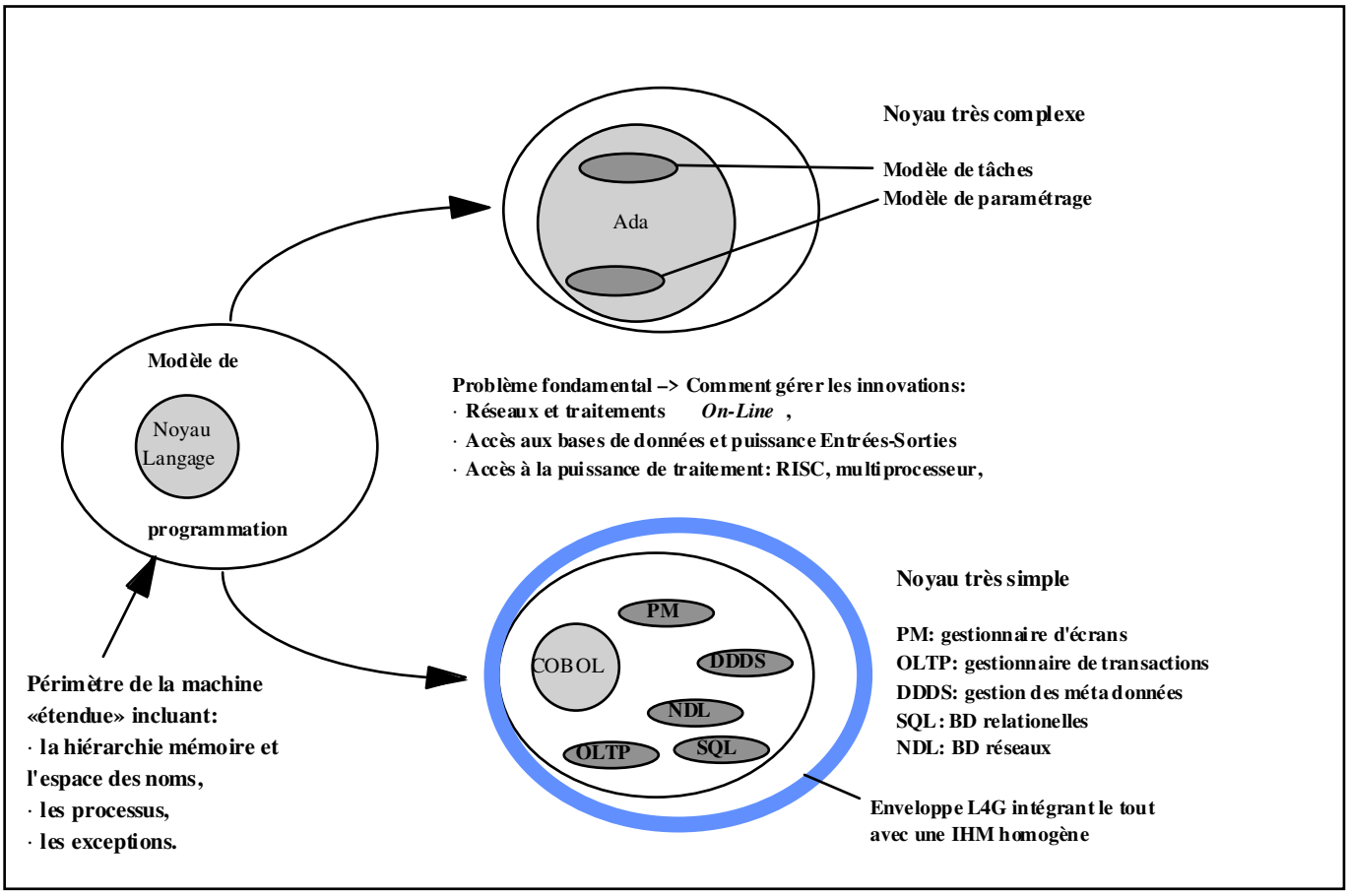
Ce dernier point pose le problème des langages de programmation, et du rôle qui est le leur: est-ce un système de signes permettant la communication entre l'homme et la machine, ou un système de signes permettant aux hommes de communiquer entre eux à propos de ce que pourrait faire la machine?

La fin d'une illusion

Ce n'est pas le lieu de faire le bilan des langages de programmation, un ouvrage complet serait nécessaire[16], mais de donner quelques éléments d'appréciation sur ce qu'il faut bien appeler la fin d'une illusion.

Sur la scène de la programmation, on a d'un côté un monde d'applications à profil système où l'on ne peut que constater l'échec relatif de Ada, à tout le moins, et le triomphe de C; et de l'autre un monde d'applications dites de gestion, les systèmes d'information et de communication, où COBOL reste l'élément stable, avec de nombreuses innovations à la périphérie.

Mis côte à côte, le contraste entre les deux approches est saisissant:



D'un côté, un monde fermé, de l'autre un monde ouvert. D'un côté un monde d'utilisateurs «moyens»[17] qui ont formidablement réussi à dominer leur problèmes, de l'autre un monde d'experts qui n'ont pas su faire la part des choses entre les besoins réels des architectes systèmes, et les besoins artificiels,

[16] On peut lire B.Walraet, *Programming, the impossible challenge* qui développe une perspective intéressante.

[17] Cf. la définition de Tsichritzis

ou supposés, générés par un comité peu représentatif des utilisateurs qu'il était sensé servir, oubliant au passage qu'un comité sert d'abord celui qui le finance!

Le plus surprenant, dans toute cette affaire, est que le comité de définition du langage Ada, sur les bases d'un cahier des charges qui se voulait exigeant — c'était la concaténation de tous les problèmes connus de l'époque! —, avait réuni un ensemble de compétences uniques, dont la somme[18] a certainement été fatale à la cohérence de la proposition, une forme nouvelle d'overdose!

Il est étrange que les deux tentatives de langages hégémoniques, la première avec le PL1, toutes deux poussées par deux organismes hégémoniques, le DoD ayant pris le relais d'IBM, se soient toutes deux terminées de la même façon. La seconde ayant, de plus, entraînée la mort de PASCAL qui était très bien armé pour jouer le rôle de COBOL dans le monde des systèmes, pour laisser finalement le champ libre à C[19] qui n'est vraiment pas ce que l'on a fait de mieux.

Une première leçon que l'on peut en tirer est qu'un comité, aussi «compétent» soit-il, n'est jamais une condition nécessaire au succès, bien qu'il puisse être une formidable machine à capter les crédits, et puisse être flatté[20] pour cette simple raison. De par sa nature, un comité est naturellement inflationniste et aura tendance à empiler, souvent correctement, les concepts, ce qui conduit à une grande complexité, et est exactement l'inverse de ce qui est souhaitable lorsque l'on a affaire à des systèmes eux-mêmes très complexes.

Ce devrait être un sujet d'interrogation qu'un langage dont il a été de bon ton de proclamer la nullité, survive ainsi à tous ses détracteurs. Ce «scandale» cache certainement quelque chose de profond. Le langage informatique doit rester neutre[21] vis-à-vis du monde réel, au contraire du langage humain qui est sémantique par essence et dont la syntaxe n'est au fond qu'une commodité, nécessaire, de communication. L'ordinateur, lui, ne comprend rien, il se contente d'exécuter!

De même que la pratique d'une langue, et la capacité à dire des choses intéressantes, n'est pas réductible à la connaissance de la grammaire, de même la maîtrise de la programmation d'un système n'est pas réductible à la maîtrise d'un langage de programmation, bien qu'elle en soit une condition nécessaire. La confusion est fréquente, et d'ailleurs commode, car elle évite de s'interroger, ou de se positionner, par rapport à la réalité ou à la pratique industrielle. Dans le domaine de la programmation, il faut scruter en permanence ce que les

---

[18] Dénoncée en son temps par N.Wirth, E.Dijkstra,…

[19] D'un coté le DoD US , et un investissement qui a largement dépassé le milliard de $, de l'autre deux ingénieurs des Bell labs.

[20] Cf. le rôle de l'ADI en France dans la décennie 80!

[21] Cf. R.Carnap *The logical structure of the world* livre remarquable et très profond. Carnap développait ce qu'il a appelé le principe de la tolérance de la syntaxe qu'il formulait ainsi: «It is not our business to set up prohibitions, but to arrive at conventions».

utilisateurs des langages[22] ont à «dire» aux systèmes sur lesquels leur production s'exécute.

Les grandes catégories d'outils

On peut considérer qu'il y a trois grandes catégories d'outils:

— ceux qui permettent la mise au point de la logique d'ensemble, et servent essentiellement aux architectes: ce sont les outils de conception qui ne sont en aucune façon une condition suffisante d'une bonne architecture.

— ceux qui permettent la programmation et la mise au point de la logique de détails: ce sont les environnements de programmation; ils sont absolument nécessaires.

— ceux qui permettent de décrire le procédé de fabrication — gestion de configuration, gestion de documentation, gestion de projet, assurance qualité — et améliorent la communication entre les individus et les groupes, car il peut bien sûr y avoir des erreurs de fabrication: ce sont les outils de gestion du processus de développement, ils sont indispensables pour les très grands projets.

La condition du succès

C'est la conjonction de ces trois catégories d'outils, dans les environnements de développement intégrés, qui rend la détection efficace.

L'expérience, fondée sur l'analyse des succès et des échecs, montre qu'un outil est d'autant plus efficace qu'il est bien adapté à sa fonction; la fonction crée l'outil, comme on dit. Dans le domaine de la programmation des systèmes, quels qu'ils soient, les notions d'états, c'est-à-dire des données, et de transitions entre les états, c'est-à-dire des traitements, sont une observation première, et une dichotomie fondamentale, sur la nature des objets informatiques, et de leur construction. La modélisation de l'information, en terme d'états-transitions, est l'objet de la théorie des machines à états finis qui est au génie logiciel, ce que la thermodynamique est à la théorie des machines thermiques, ou l'électromagnétisme à l'électrotechnique. C'est dire son importance! La théorie des automates réalise le compromis rare d'être à la fois parfaitement bien fondée, et non triviale, au plan mathématique et utile à l'ingénieur[23], comme le sont le calcul des probabilités, ou le calcul différentiel et intégral, dans d'autres domaines. L'un des objectifs premiers des langages de programmation doit donc être la représentation et la manipulation des automates nécessaires à la description des systèmes.

Compenser les erreurs: l'approche statistique

---

[22] Cf. R.Thom qui fait écho à D.Hilbert: «Tous les grands progrès théoriques proviennent de la capacité des inventeurs à se *mettre dans la peau des choses*, pour pouvoir s'identifier par empathie à n'importe quelle entité du monde extérieur» dans *Prédire n'est pas expliquer*.

[23] Il est remarquable que cette théorie soit le fondement de deux domaines aussi dissemblables en apparence que l'ingénierie des compilateurs, et l'ingénierie des protocoles de télécommunications. On notera la parfaite neutralité de la théorie.

Lorsque le processus de détection des erreurs est correctement conduit, à l'aide de revues d'architecture, d'inspections de documents, de tests, de preuves dans certains cas très favorables, puis d'utilisation *in situ* du logiciel, on constate une relative stabilisation du taux d'erreurs. Généralement reproductibles, en début de vie du logiciel, les contextes d'apparition des erreurs deviennent progressivement non-reproductibles, d'où l'appellation plaisante *d'Heisen-bug* donnée à cette catégorie d'erreurs. Le contexte d'apparition de l'erreur devient trop fugace pour permettre une prise d'information, suffisamment précise, nécessaire au diagnostic: il n'est plus possible de localiser précisément l'erreur! Ce genre d'erreurs est caractéristique de situations où il y a des conflits de ressources et, d'une façon plus générale indéterminisme local. Tout ceci suggère une approche plus statistique de la vérité, car là comme ailleurs dans notre monde imparfait, l'absolu n'a pas sa place. Cette problématique était d'ailleurs celle de Von Neumann[24], dès l'origine.

Aucun domaine de l'activité humaine n'est à l'abris de l'erreur, ce qui ne nous empêche pas de vivre. Tout dépend de l'évaluation du risque associé à la présence des erreurs, et de la confiance que l'on a dans les mécanismes et les procédures qui vont permettre de compenser les défaillances occasionnées par les erreurs. Dans un milieu présentant des défauts, le schéma descriptif n'est plus:

> **SI** A **ET** B se produisent, **ALORS** C se produira

mais:

> **SI** A **ET** B se produisent, **ALORS**
>
> **éventualité 1**: C se produira avec la probabilité P_1,
>
> **éventualité 2**: D se produira avec la probabilité P_2,
>
> etc.
>
> avec $P_1 > P_2 > \ldots > P_n$

C'est-à-dire qu'à chaque situation est associée un éventail d'alternatives dont chacune a une certaine probabilité de survenue. L'enchaînement des situations peut être alors décrit à l'aide d'une matrice de probabilités qui nous ramène à la loi de la variété indispensable de W.R.Ashby[25] qui est une variante qualitative très intéressante du deuxième théorème de Shannon.

Il n'y a aucune raison d'imaginer qu'en matière de programmation on puisse atteindre une vérité absolue, ou de déterminisme total, mais que, comme dans toutes les autres sciences de l'ingénieur, on doive se contenter de probabilités et de vérités statistiques.

Most bodies of knowledge give errors a secondary role, and recognize their existence only in the later stages of design. Both coding and information

---

[24] Cf Oeuvres complètes, Vol. 5 *Probabilistic logics and the synthesis of reliable organisms from unreliable components*, et aussi *Theory of self-reproducing automata*.

[25] Cf. *An introduction to cybernetics*, Chap. 7 et 8.

theory, however, give a central role to errors (noise) and are therefore of special interest, since in real-life noise is everywhere.[26]

On ne saurait mieux dire!

[26] Cf. R.W.Hamming, *Coding and information theory*.

La sûreté de fonctionnement du logiciel

Le développement de l'interface entre les hommes et les systèmes informatiques, on dira systèmes d'information et de communication pour être plus précis, augmente ce que l'on peut appeler la surface de contact entre les usagers que nous sommes tous, et les systèmes. Deux phénomènes nouveaux prennent une importance considérable:

- L'usager s'attend à trouver, dans le système, un certain nombre de fonctions et/ou de mécanismes qui lui paraissent évidents compte tenu de son modèle psycho-cognitif.
- L'usager s'attend, en cas de mauvaise manipulation, d'altération de l'environnement nominal, à une réaction du système semblable à celle qu'il aurait lui-même dans une situation à risques; c'est-à-dire qu'il restaure les conditions nominales et rétablisse un contexte de travail cohérent.

A vouloir faire des systèmes étroitement imbriqués à l'activité humaine, il devient nécessaire de doter les systèmes de comportements compatibles avec les comportements humains[27]. Ces comportements sont caractérisés par:

- une extraordinaire capacité d'adaptation des hommes à la nouveauté et au développement des compétences grâce à l'apprentissage, et aux feed-backs positifs (renforcement) dont il est capable.
- une non moins extraordinaire capacité à survivre dans des environnements extrêmement hostiles, qui est également une forme d'adaptation à des situations qui requièrent des feed-backs négatifs (annulation).

En terme d'ingénierie, il s'agit là de deux formidables défis que l'on peut formuler ainsi:

[27] Cf. J.Piaget, *L'équilibration des structures cognitives*, texte remarquable sur l'apprentissage.

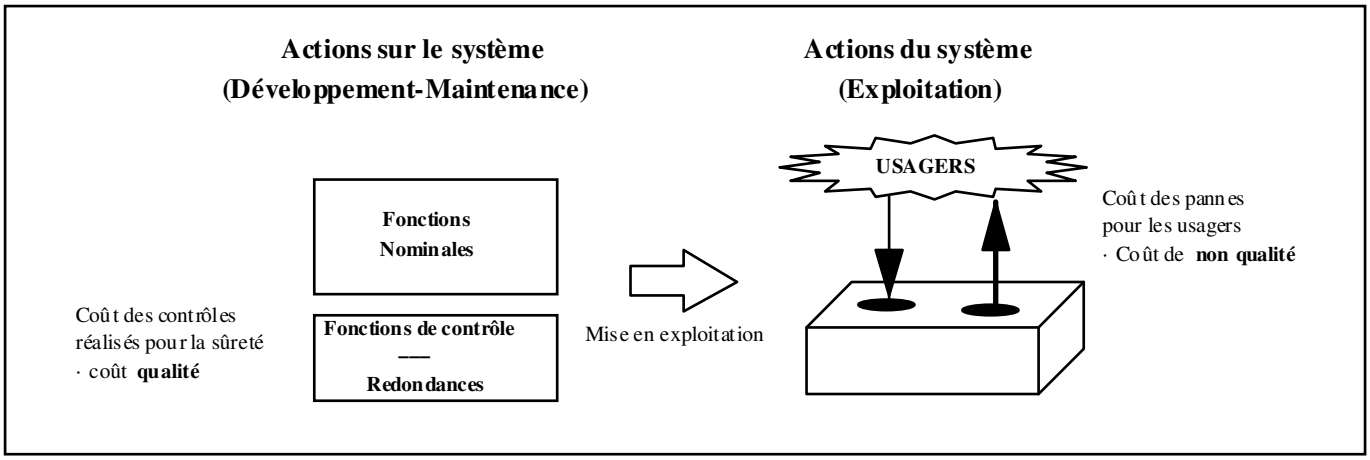
Premier défi

Comment intégrer, dans un système en cours de développement et/ou d'exploitation, des nouveautés au même rythme que celles-ci apparaissent, c'est-à-dire avec des délais assez courts, et ceci sans nuire à la qualité de services et/ou aux fonctions antérieurement acquises.

Second défi

Quelle est la nature, la densité, la fréquence d'appel des mécanismes réparateurs, ainsi que le niveau des ressources qu'il faut consigner, pour garantir une disponibilité du système fixée à l'avance, en fonction du contexte d'emploi.

Il est clair que cette adaptabilité, positive et/ou négative, à l'environnement va avoir un coût qui est le prix à payer pour l'obtention de cette propriété, par définition très diffuse.

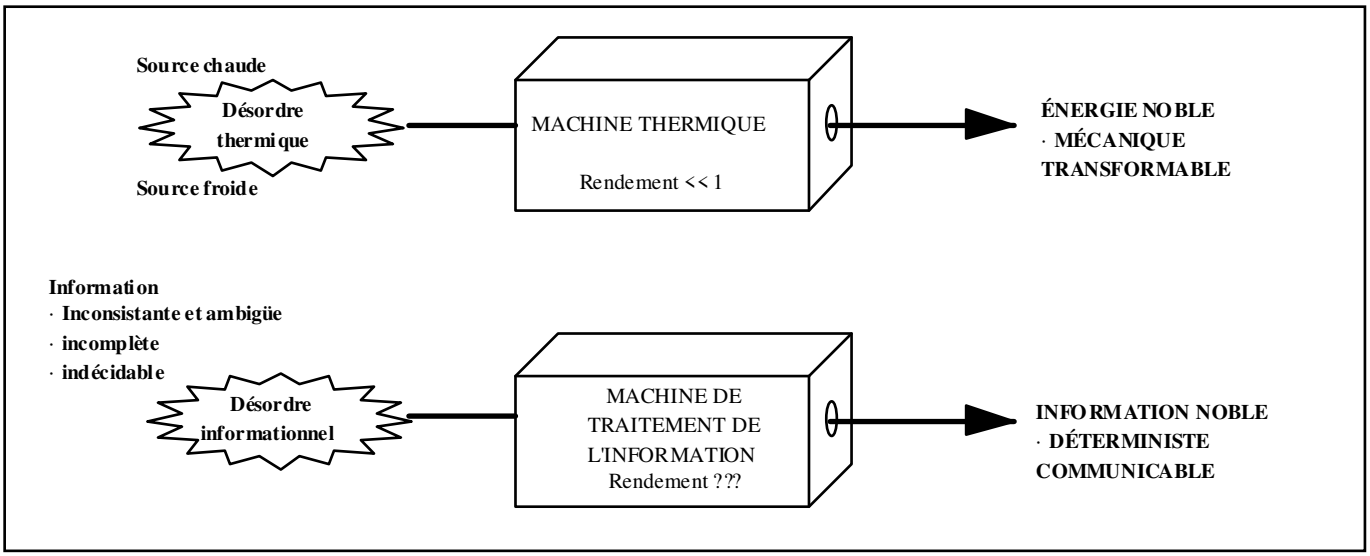


Une analogie thermodynamique

La dualité information/entropie a été une source d'inspiration constante de la pensée des «pères fondateurs» des machines à calculer[28], comme on disait alors. Leo Szilard, démystificateur du paradoxe du démon de Maxwell, compatriote de John von Neumann, engagé comme lui dans le projet Manhattan y était sans doute pour quelque chose.

La thermodynamique, fixe les conditions macroscopiques dans lesquelles un système fermé, quelconque, échange et transforme les différentes formes de l'énergie. Le rendement de ces transformations est intrinsèquement limité par le second principe, et la chaleur n'est jamais récupérable dans son intégralité.

[28] Von Neumann et ses collègues parlaient de «computing instrument»!



On sait que le rendement d'une machine thermique est d'autant meilleur, bien que de toute façon faible, que les états successifs du système parcourent un cycle, le cycle de Clausius, entre une source chaude et une source froide. On récupère ainsi une partie du désordre thermique sous une forme organisée et aisément transformable.

Le carburant utilisé par les systèmes informatiques est double:

- C'est d'une part un effort humain qui sert à développer et exploiter le système.
- C'est d'autre part un effort «machine» qui sert à exécuter les traitements programmés dans le système. Cet effort nécessite des instructions[29], de la mémoire et des entrées-sorties.

Un ingénieur doit impérativement s'intéresser aux conditions économiques qui régissent les conditions de transformations de ces différents efforts, et connaître en particulier les limitations, si tant est qu'il en existe, de façon à ne pas s'engager dans des fabrications irréalisables.

La forme noble de l'information, celle qui est utile à l'homme, est l'information déterministe: si un ensemble de conditions sont réunies, on sait exactement prédire le résultat. La régularité, le caractère reproductible de l'opération est une condition absolue d'utilité. Sans régularité, sans répétition, pas d'apprentissage possible, donc pas de communications possibles. L'étude du rendement de la transformation est donc ramené à l'étude des conditions qui permettent au système d'avoir un comportement déterministe.

Entropie et information sont des grandeurs de même nature, bien que l'une soit une énergie et l'autre un nombre pur. La quantité d'information attachée à un événement est d'autant plus grande que cet événement est rare, c'est-à-dire imprévisible. Dans un univers «organisé», déterministe, la quantité d'information est faible puisque tout y est prévisible. C'est la source froide. L'équilibre de cette source nécessite un apport constant d'énergie; elle se

[29] Généralement comptées en MIPS, qui est en fait une puissance; on pourrait parler de MIPSheure.

comporte comme un système dissipatif[30]. La perte d'information qui accompagne le traitement de l'information peut être montrée de façon très simple:

Si l'on considère le processus "SOMME DE DEUX NOMBRES", ce processus établit une règle de correspondance entre un ensemble de couples (a,b) et de résultats (r). À l'événement résultat (5) va correspondre les 6 couples (0,5) (1,4) (2,3) (3,2) (4,1) (5,0). Dans l'univers des entiers, la probabilité d'occurrence de l'événement (5) est donc 6 fois celle de l'un quelconque des couples qui lui ont donné naissance, la quantité d'information[31] qui lui est attachée est donc $Log_2 6$ fois plus faible, ce qui caractérise la perte.

Si l'on veut que la transformation soit réversible, il faut garder trace de l'un quelconque des arguments du couple, moyennant quoi, et compte tenu de l'existence d'une opération inverse, on saura revenir à l'état initial. Soit donc le schéma:

$$(a,b) \xrightarrow{S} (r)$$

qui devient:

$$(a,b) \xrightarrow{S_R} (r,a) \xrightarrow{S_R^{-1}} (a,b)$$

Sur cet exemple simple, on voit que le coût de la réversibilité se traduit par:

— stockage supplémentaire pour l'élément (a)

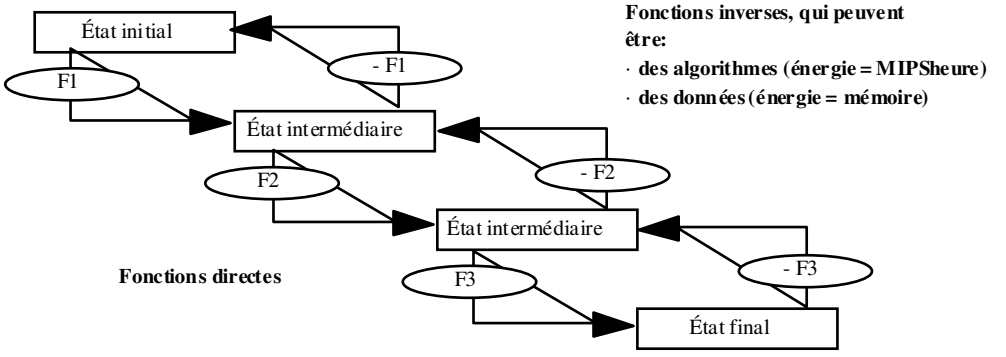
— programmation et utilisation de l'opération inverse.

Ces éléments sont donc une redondance, au sens de la théorie de l'information, et un apport énergétique dans le système dissipatif.

Ce schéma volontairement très simplifié, suggère un schéma général beaucoup plus proche de la réalité:

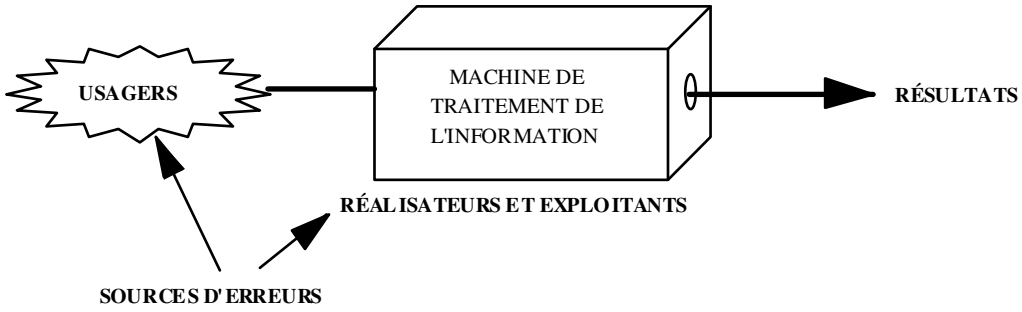
[30] Cf. I.Prigogine, *La nouvelle alliance*.

[31] Généralement donnée en Log à base 2, ce qui donne le nombre moyen de questions à poser pour trouver la bonne éventualité.



Les états intermédiaires forment des langages «cachés»

Dans un univers sans erreur, seule importe l'obtention du résultat et les fonctions directes qui permettent de l'obtenir. Dans notre univers imparfait, baigné d'erreurs, il est essentiel de discriminer les causes des erreurs.



En l'absence des fonctions inverses, le diagnostic de la défaillance sera un processus extrêmement coûteux car la cardinalité de l'ensemble des cas possibles pouvant conduire à la situation défaillante est très grande[32]. Parmi ces cas, seuls quelques-uns seront véritablement intéressants. Le rôle des fonctions inverses, et la bonne organisation des fonctions directes, c'est-à-dire l'architecture du processus de calcul, a comme objectif premier de faire baisser la cardinalité des ensembles dans lesquels il faut rechercher la présence d'états erronés, et conséquemment de faire baisser le coût de cette recherche. On peut donc dresser un bilan «énergétique» de cette recherche comme suit:

Énergie totale =

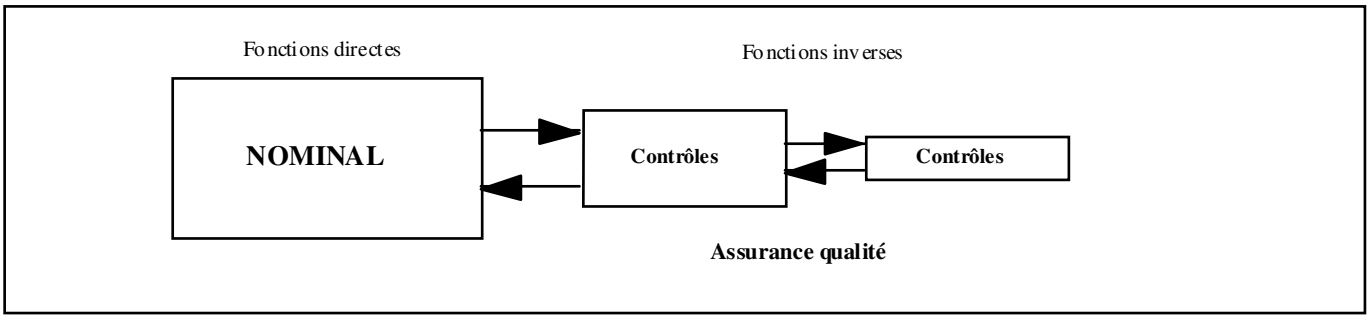
— énergie «utile», c'est-à-dire les fonctions directes,

— énergie de régulation et/ou compensation, c'est-à-dire:

[32] Pour une fonction linéaire à *N* places, il y a $O(r^{N-1})$ n-uplet d'arguments possibles pour une valeur r; pour un tri de *N* éléments, il y a *N*! configurations possibles.

- contrôles à mettre en oeuvre au niveau du processus de développement, c'est-à-dire nombre d'hommes-mois pour les inspections, les revues, etc.
- conception et réalisation des fonctions inverses utilisées lors de l'exécution, c'est-à-dire hommes-mois de développement qui, en plus, consommeront des ressources machine.

L'«énergie» de régulation/compensation est directement liée au niveau de perturbation/agression (i.e. le bruit) auquel sera soumis le système, c'est-à-dire: soit environnement très perturbé, soit personnel travaillant avec un fort taux d'erreurs. En terme de coût, cette énergie est une assurance. D'où une architecture de contrôle:



L'optimisme naturel, ou le refus de la réalité, conduit souvent à négliger la part de l'énergie qu'il faut réserver pour le contrôle (c'est-à-dire l'existence d'une source froide, qui est un centre organisateur!) avec les conséquences prévisibles que l'on sait.

L'analogie avec le 2ème théorème de Shannon est totale car ce théorème garanti l'existence d'un codage qui va exactement compenser le bruit auquel est soumis le canal de transmission, pour autant qu'on connaisse le niveau de ce bruit. Ce théorème existentiel, rarissime dans les sciences de l'ingénieur, ne donne pas l'algorithme de construction d'un tel codage qu'il faut aller chercher dans les codes redondants de Hamming, Huffman, etc.

On en restera là avec cette analogie qui pourrait alimenter quelques thèses intéressantes.

L'homme et la complexité

On sait que le processus de calcul mis en oeuvre pour résoudre un problème est intrinsèquement limité par la complexité algorithmique attaché à ce problème. C'est ainsi que l'algorithme de jointure du modèle relationnel, qui permet de sélectionner les couples, pris dans deux ensembles, partageant une même propriété requiert $N1 \infty N2$ comparaisons, pour parcourir exactement le produit cartésien des deux ensembles. On dit que cette opération est en $O(N^2)$.

De même, certains algorithmes requièrent un volume de mémoire, et/ou un débit d'entrées-sorties qui augmentent en fonction du nombre d'éléments manipulés.

Ces fonctions de complexité sont une limitation fondamentale des traitements susceptibles d'être effectués sur une machine dont certains éléments, en

particulier le débit des bus de données, limitent radicalement la complexité des problèmes susceptibles de leur être soumis, en particulier si il existe des contraintes de temps de réponse. Une simulation n'a d'intérêt que si elle s'effectue plus rapidement que le phénomène simulé!

Un calcul élémentaire illustre parfaitement l'ampleur de cette difficulté. Si l'on reprend l'exemple du produit cartésien dans le cas simple ou $N_1=N_2=1000$ et où la comparaison nécessite 50 instructions en moyenne, l'ensemble du calcul nécessite donc $50\infty10^6$ instructions. Si l'on prend l'hypothèse simplificatrice de 1 mot mémoire[33] lu et/ou écrit par opération, il faudra acheminer $200\infty10^6$ octets dans l'unité centrale, soit une bande passante de $1{,}6\infty10^9$ bits[34], sans compter les bits de contrôle (+25%), soit au moins 2 GigaBits à transférer.

La performance humaine

Il se trouve que si l'homme effectue certaines opérations beaucoup moins vite que l'ordinateur, il en effectue certaines autres à une vitesse incomparablement supérieure, en particulier tout ce qui est approximatif, flou, imprécis, etc. — sans parler de l'apprentissage— mais suffisant tout de même pour prendre une décision immédiate. C'est la fonction de *bon sens* et d'appréciation des situations.

Il faut donc faire extrêmement attention lorsque l'on répartit les rôles entre l'homme et l'ordinateur.

Certaines opérations d'apparence anodine, peuvent requérir un très grand volume de programmation, et des algorithmes de très grande complexité. Une opération d'optimisation, sur un ensemble comportant N éléments, nécessite presque toujours la manipulation de l'ensemble des parties dont la cardinalité est 2^{N-1}, auquel va correspondre un ensemble fonctionnel de cardinalité $2^{2^{N-1}}$. C'est souvent l'origine de l'explosion combinatoire.

Le canal de communication entre l'homme et la machine

Idéalement, on peut donc assimiler ce transfert d'opérations de l'homme vers la machine sous la forme de canaux d'entrées supplémentaires E, et se poser la question du volume V de ressources (hommes-mois, UC, Mémoire, E/S) nécessaires pour les exécuter.

En d'autres termes, on cherche à établir une relation du type $V=F(E)$.

La nature de notre monde physique permet l'établissement de nombreuses relations de ce type. Ces relations sont connues sous le nom de principe de similitude[35], et on les retrouve dans les équations de dimensions de la physique, historiquement attestées[36] dans l'oeuvre de Fourrier, et peut-être même avant. Ces principes ont été exploités pendant de longues années par les

---

[33] Dans les machines récentes, la largeur de mot est plutôt 64 bits.

[34] NB: pour mémoire, le débit théorique d'Ethernet = 10^7 bits par seconde.

[35] Cf. D'Arcy Thompson, *On growth and form* et L. von Bertalanffy, *Théorie générale des systèmes*.

[36] Cf. Joseph Fourier, *Théorie analytique de la chaleur*, 1822.

sciences de l'ingénieur[37], pour passer d'une maquette de réalisation à l'objet réel, avant que les progrès de la simulation ne rendent ce genre de considérations dépassés pour des objets du monde physique.

Il n'y a pas, à ce jour, d'études sérieuses[38] sur le sujet pour ce qui concerne les systèmes informatiques, mais tout laisse à penser que cette relation est de nature chaotique[39], ce qui pourrait expliquer le caractère aléatoire de toute prédiction du volume d'un système autrement qu'à partir de la réalisation d'un prototype.

Dans la mesure où le nombre d'erreurs résiduelles est une fonction monotone croissante de la longueur du texte — une autre relation de similitude intéressante — on a tout intérêt à bien connaître l'amplitude des variations, mais surtout à faire un texte court! ce qui est une règle empirique mise en oeuvre par tous les chefs de projets expérimentés.

Les principes de la détection des défaillances

Le manque de ressources des premières générations d'ordinateurs a créé une culture de programmation fondée sur la recherche du codage des données et des algorithmes le plus compact possible. À la jauge de l'échelle de complexité évoquée précédemment, nos ordinateurs modernes sont encore très limités, et surtout ils fragmentent la ressource de calcul dont on reparlera plus loin.

Cette attitude fait immanquablement penser au 1er théorème de Shannon, sur l'existence de codes économiques[40], fondés sur la probabilité d'occurrence et la fréquence d'apparition des éléments à coder.

Ces codes «économiques» présentent de graves défectuosités du point de vue de la fiabilité, car un élément du code a toujours un sens. Si le code résultant d'un calcul est erroné, le processus continue de s'exécuter jusqu'à la détection d'une incohérence sémantique, soit par le programme, soit, en désespoir de cause et donc très tard, par l'usager du système. Le temps de latence entre l'occurrence du défaut et sa détection peut donc être très grand, et occasionner de très graves dégradations de service.

Le seul remède, face à ce type de situations, est de désaturer au maximum le code initial de façon à utiliser la syntaxe là où elle est extraordinairement efficace, c'est-à-dire la reconnaissance de séquences valides parmi toutes les séquences possibles du monoïde associé au code. Soit la situation suivante:

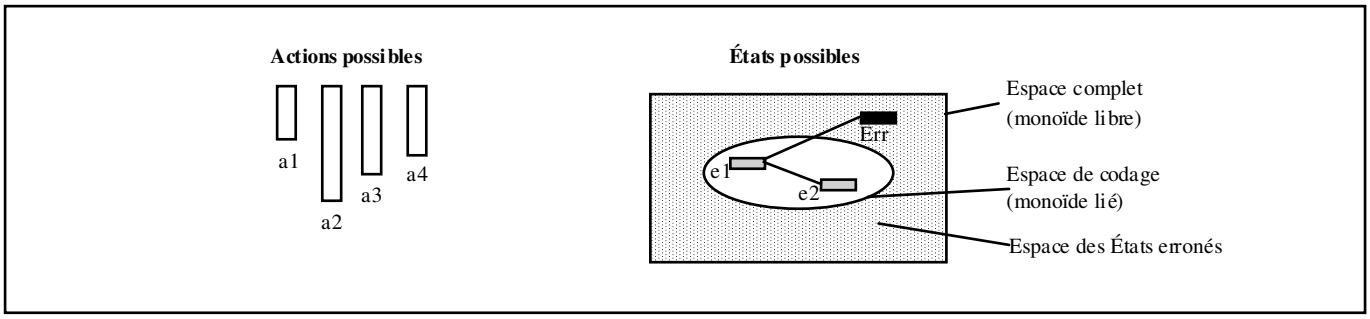
---

[37] C'était encore enseigné à l'École Centrale dans les années 60.

[38] Ce qui s'en rapproche le plus est la méthode des *Function points*, due à A.J.Albrecht d'IBM.

[39] Cf. G.Chaitin, *Algorithmic information theory*.

[40] Cf. C.Shannon, W.Weaver, *The mathematical theory of communication*.



Les actions a_1, a_2,…, a_n appliquées aux états possibles de l'espace de codage les transforment jusqu'à obtention de l'état final (c'est donc un automate). Si l'une des actions est erronée, il existe certes une probabilité de retomber sur un code valide, mais cette probabilité sera nécessairement inférieure à celle que l'on aurait avec un code saturé, car elle nécessite la conjonction de plusieurs événements aléatoires indépendants (c'est le phénomène des double pannes, et au delà) qui est l'essence même du 2ème théorème de Shannon.

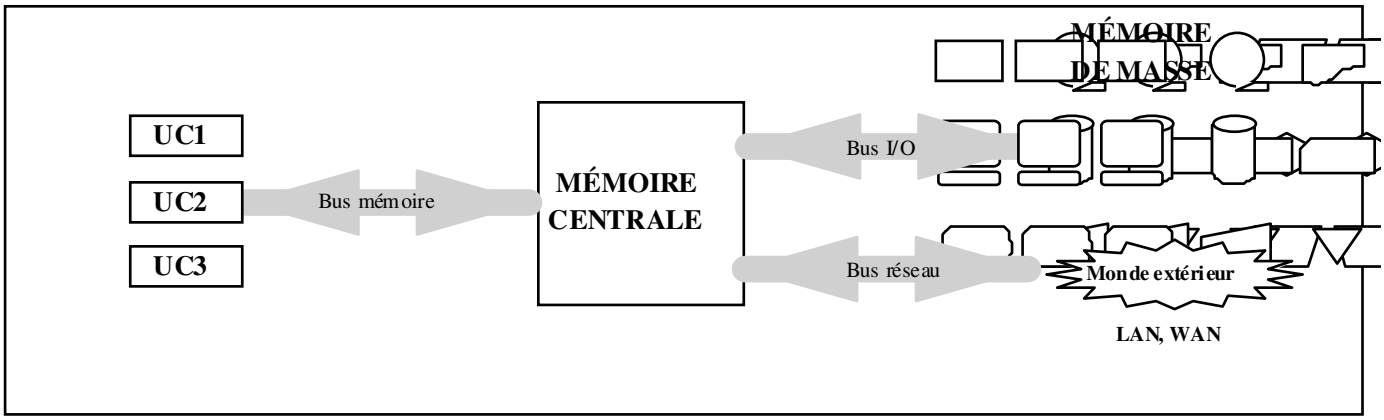
Si l'espace de codage est suffisamment peu dense, on peut espérer, à l'aide de scrutations périodiques et de vérifications de la cohérence des codes, détecter suffisamment vite l'apparition des défaillances pour pouvoir diagnostiquer l'action erronée, et appliquer les fonctions inverses pour redémarrer dans des conditions qui restaurent le postulat du déterminisme. De cette façon, on a une claire appréciation de l'effort à consacrer à la sûreté:

- Plus la *cardinalité* de l'espace des états à surveiller est grande, plus le volume des fonctions inverses sera important, ce qui nécessitera un certain nombre d'hommes-mois de développement,
- Plus la *fréquence* de la scrutation est grande, plus le nombre de MIPS, mémoire, E/S consacrées à cette scrutation sera important.

En l'absence de cette source froide, qui n'est que le prix à payer des incertitudes de l'environnement et de nos propres turpitudes, le rendement global de l'effort sera nécessairement très bas, comme on le constate dans de nombreux systèmes qui connaissent de graves difficultés.

Les nouvelles architectures

On a vu précédemment que l'ordinateur reste terriblement limité face à des problèmes de complexité et de taille moyennes, en particulier par le volume d'information que ses organes d'entrées-sorties sont capables de transférer.



Ce fait était bien connu dès l'origine des premiers ordinateurs, et John von Neumann avait d'ailleurs élaboré une première théorie du parallélisme pour lever, au plan logique, ces limitations dont il était parfaitement conscient: la théorie des automates cellulaires. L'idée, simple, étant qu'un problème de grande taille peut être fragmenté en un ensemble de problèmes plus réduits, susceptibles de s'exécuter en parallèle sur un ensemble de calculateurs organisés de façon ad hoc.

Un ordinateur vectoriel, ramène la complexité d'un produit vectoriel qui est en $O(N^2)$, à une complexité apparente de l'ordre $O(N)$, d'où leur intérêt.

Le formidable développement de la microélectronique et des technologies de la communication, comme la fibre optique, permettent la réalisation d'ensembles interconnectés, inimaginables à l'époque de von Neumann. Le rêve logique est donc devenu réalité.

Deux grandes familles d'interconnexion sont actuellement à l'essai:

- l'interconnexion telle qu'on la voit dans les machines dites massivement parallèles, dont l'un des prototypes les plus avancés est la *Connection-Machine*, qui mettent en oeuvre un couplage très étroit entre des milliers d'éléments actifs.
- l'interconnexion qui consiste à fédérer différentes machines, a priori homogènes, au moyen de réseaux locaux à haut-débit de façon à spécialiser fonctionnellement les machines, à mettre en commun des ressources comme les bases de données; c'est l'approche *Cluster*, initialement proposée par DEC.

À l'intérieur de ces familles, et entre elles, on trouve tout un ensemble de variantes possibles, mais le problème est toujours le même; il s'agit de savoir si:

— $N1$ UC de puissance p sont équivalents à 1 UC de puissance $N1 \infty p$,

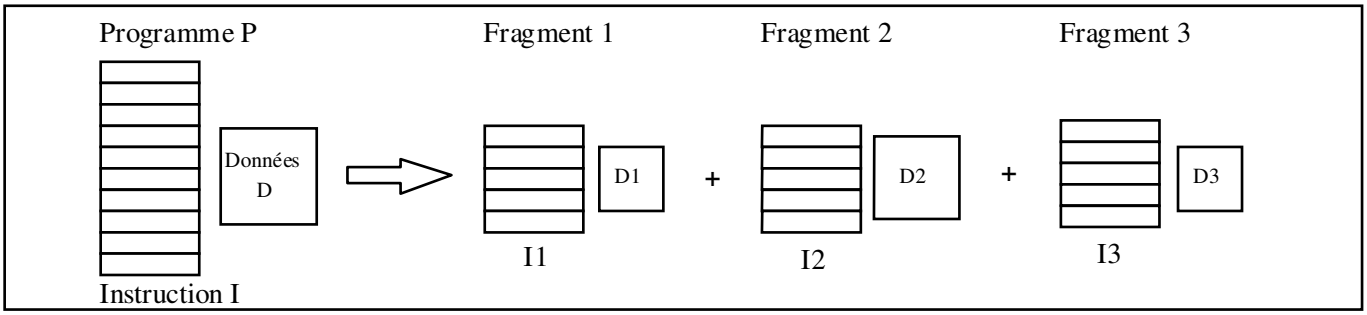
— $N2$ mémoire de taille s sont équivalentes à 1 mémoire de taille $N2 \infty s$,

— $N3$ bus de débit d sont équivalents à 1 bus de débit $N3 \infty d$,

tout ceci en cachant aux programmeurs de ces nouvelles machines tout le détail des structures internes de façon à conserver ce qui est un acquis essentiel du génie logiciel: l'indépendance — relative — de la formulation d'un problème par rapport à l'architecture sous-jacente.

Le coût de la fragmentation

On peut schématiser le processus de fragmentation d'un programme comme suit:



La puissance n'est additive que si les fragments sont indépendants, c'est-à-dire que:

— les données sont disjointes (on pourrait admettre une réplication du code car c'est un invariant, mais pas une réplication des données),

— il n'y a pas de dépendances fonctionnelles entre les données (l'une n'est pas le résultat de l'autre, ce qui induirait une relation d'ordre).

Cette propriété est très facile à obtenir si les fragments proviennent de programmes différents. C'est ce qui a été exploité dès les années 60 dans les systèmes d'exploitation à multiprogrammation, puis amélioré avec les machines multiprocesseurs et, bien sûr, dans les *Clusters*.

Cette propriété est très difficile à obtenir si il s'agit du même programme.

On peut concevoir que par scrutation conjointe des instructions de P et de ses données D, on puisse faire apparaître des fragments parallélisables, mais ce problème est de complexité exponentielle, donc impraticable avec des programmes de taille intéressante. On peut imaginer qu'un programmeur talentueux[41] puisse faire apparaître des structures intéressantes, car le cerveau humain est plus à même de résoudre ce genre de problème, même de façon approximative. Cette recherche est fondée sur les données dont on voit, une nouvelle fois, l'importance primordiale, et leur préexistence, par rapport aux instructions. C'est la structure de données qui engendre la structure des traitements, et non l'inverse. Toute banalisation des données, par rapport aux traitements, ne peut rendre l'analyse de ces problèmes que plus difficile.

À supposer d'ailleurs que les données et les instructions aient été restructurées automatiquement, il faut que l'objet résultant reste compréhensible à son auteur, car compte tenu du problème N°1, le programme contiendra toujours des erreurs résiduelles qu'il faudra corriger, donc rester capable d'analyser les défaillances sur une structure qui peut être très différente de la structure initiale.

Enfin, dans le cas d'une évolution, ce qui est le destin de tout programme présentant un intérêt, il faudra s'assurer que les ajouts sont compatibles avec la fragmentation initialement obtenue.

Cette situation est particulièrement difficile dans le cas des ordinateurs massivement parallèles, à réseau d'interconnexion programmable, comme la *Connection-Machine*. La fragmentation, et l'interconnexion, sont à la charge d'un compilateur d'un nouveau type. La mise au point des programmes correspondants nécessite un *debugger* non intrusif, et particulièrement furtif, qui respecte la temporalité des programmes pour ne pas en altérer les conditions d'exécution, et masquer ainsi les pannes de synchronisation.

Si le programme ne présente pas de grandes régularités, aisément détectables, il est peu probable que l'on puisse jamais utiliser la pleine puissance de ces architectures. Or il semble que beaucoup de problèmes «intéressants» aient une structure chaotique.

De plus, on se heurte à une caractéristique incontournable de la pensée humaine consciente, qui est de devoir s'énoncer de façon séquentielle, pour pouvoir s'exprimer et communiquer à travers nos langues naturelles. Nos langages artificiels n'ont aucune raison d'échapper à cette contrainte fondamentale. L'une des difficultés majeures de la programmation concurrente tient à notre incapacité à pouvoir penser simultanément plus de 3 ou 4 flots d'instructions s'exécutant concurremment. Alors, quand il y en a plusieurs milliers! et ceci indépendamment du fait que certains problèmes ne sont pas fragmentables.

Un autre effet, singulièrement pervers de la fragmentation, est la délocalisation de certaines erreurs, en particulier toutes celles qui vont avoir trait à la concurrence et à la synchronisation, qui au lieu d'apparaître dans un seul flot

[41] Lors d'une visite à Thinking Machines Corp, j'ai pu constater de visu que tous les programmeurs de la CM étaient des Ph.D! La société a fait faillite en 1994.

vont pouvoir apparaître dans chacun des micro-flots et dans leur combinaisons respectives qui croît comme l'ensemble des parties, ce qui correspond d'ailleurs à une augmentation de la quantité d'information du programme.

Gérer la complexité architecturale

L'une des très grandes réussites du génie logiciel a été de permettre une évolution considérable des structures matérielles de façon complètement transparentes aux usagers des systèmes. On peut donner deux exemples:

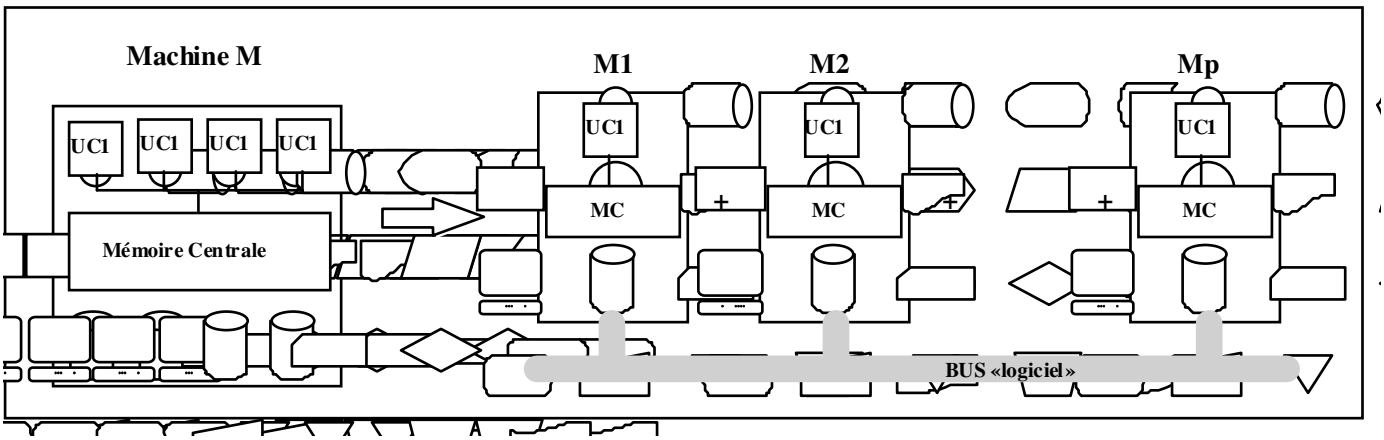
- L'architecture fine de l'unité centrale et de la mémoire, complètement cachée par les compilateurs des langages de haut-niveau (Cf. allocation de registres, gestion des caches, *pipe-line* d'instructions, etc.).
- L'accès concurrent aux bases de données, à l'aide de moniteurs de transactions qui donnent au programmeur l'illusion qu'il est seul, alors que dans la réalité il pourra y avoir des centaines d'accès simultanés.

L'énorme succès de la programmation transactionnelle, qui est l'outil quotidien de milliers de programmeurs de système d'information, est l'illustration la plus parfaite du principe de Tsichritzis rappelé dans l'introduction. Elle est à mettre au crédit des architectes des systèmes d'exploitation et de SGBD.

Ceci n'est bien sûr possible que si l'on a une claire conscience de ce que l'on peut raisonnablement demander à l'usager du système et aux différentes couches architecturales. Il peut être tentant, pour des raisons économiques et/ou techniques mal comprises de faire gérer les problèmes d'une couche par les couches voisines de celle où ils devraient être naturellement traités. On constate ce genre de discours avec les technologies réseaux (Cf. l'ATM), et/ou le Client-Serveur où la tentation semble forte d'exporter certaines difficultés vers les couches applicatives. En termes économiques, si c'est un jeu à somme positive, le transfert est intéressant, si la somme est nulle, ça ne sert à rien, si elle est négative, c'est un non-sens coûteux.

Le cas des architectures clients-serveurs est particulièrement symptomatique de cette tendance.

En raisonnant sur le prix du matériel, et en supposant que les MIPS, le débit d'E/S et la mémoire sont additifs, on peut «démontrer» qu'un réseau de serveurs-stations est beaucoup moins coûteux que son équivalent *main-frame*.



La fragmentation de M en{M_1, M_2,…, M_p} fait apparaître un objet nouveau sur la table de l'usager, appelons-le *Bus logiciel*, connu comme l'un des objets système les plus complexes à gérer, mais qui était complètement caché dans la machine M. L'usager doit désormais gérer:

- l'équilibrage de charge entre {M_1, M_2,…, M_p} et la saturation,
- l'administration,
- la sûreté de fonctionnement et la non-propagation des erreurs,
- la sécurité (confidentialité, intégrité) des informations circulant sur le bus.

La situation empire si les machines sont hétérogènes.

Cette situation est un véritable cas d'école, presque caricatural, dans lequel on a opéré un véritable transfert de complexité système vers des usagers qui n'y ont vu que du feu! Il faut espérer que ces derniers auront un bon administrateur système, et de bons programmeurs, pour rester maître de la situation.

La montée de l'indéterminisme

Les premiers ordinateurs ont été l'exemple quasi parfait de la célèbre formule de Laplace, souvent prise comme le postulat fondamental du déterminisme: «Nous devons donc envisager l'état présent de l'univers comme l'effet de son état antérieur, et comme la cause de ce qui va suivre. Une intelligence qui pour un instant donné connaîtrait toutes les forces dont la nature est animée et la situation respective des êtres qui la composent, si d'ailleurs elle était assez vaste pour soumettre ces données à l'analyse, embrasserait dans la même formule les mouvements des plus grands corps de l'univers et ceux du plus léger atome: rien ne serait incertain pour elle, et l'avenir, comme le passé, serait présent à ses yeux.»[42]. Dans son univers restreint, l'ordinateur réalisait cet idéal.

Si certains systèmes *Temps-Réel* conservent cette exigence, on s'est vite aperçu qu'une certaine flexibilité permettait de déplacer l'équilibre économique en faveur de l'usager. Des dispositifs comme les caches de données et d'instructions, les *pipe-lines*, sont des accélérateurs qui font que, sous certaines circonstances, le temps moyen d'exécution des instruction va fluctuer entre [**T1**, le plus favorable; $N \infty$**T1**, le moins favorable] avec N pouvant atteindre 10 ou plus. C'est le rôle des compilateurs de s'assurer que données et instructions sont agencées de façon telle que les programmes puissent bénéficier de ce facteur d'accélération. Ce faisant on ne sait plus assigner une durée déterminée à une séquence d'instructions, et encore moins à un programme complet. On ne connaît plus que des comportements moyens.

Cette tendance s'est amplifiée lorsque l'on a considéré qu'il était parfois plus coûteux d'assigner statiquement toutes les ressources nécessaires à l'exécution que de les allouer dynamiquement, à la demande. C'est ainsi que les dispositifs à mémoire virtuelle ont pu donner l'impression trompeuse, mais singulièrement commode, que chaque programme disposait de ressources mémoire potentiellement infinies. C'était extraordinairement astucieux, compte tenu de la pénurie de ressources des machines d'alors.

---

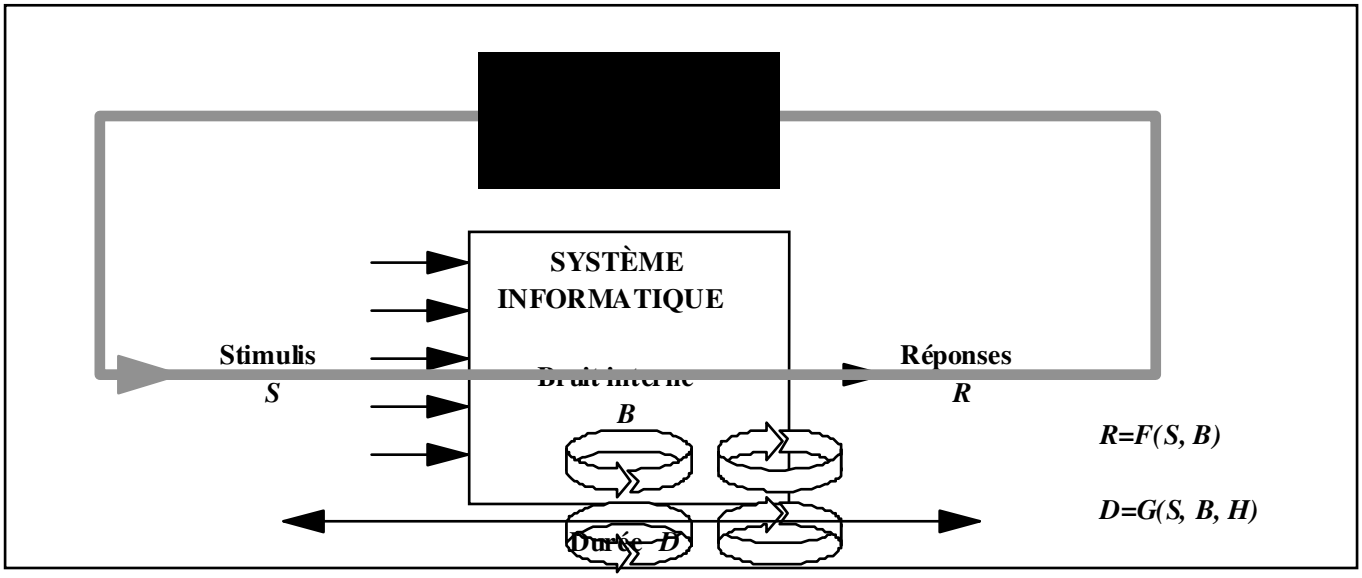
[42] Pierre-Simon Laplace, *Calcul des probabilités*, qui contient *l'essai philosophique sur les probabilités*.

Ce faisant, on a accru le niveau d'indéterminisme d'un ordre de grandeur, et rendu possible des phénomènes de saturation se traduisant par des chutes brutales des performances de la machine qui ne fait plus que gérer les conflits d'accès aux ressources, en ne laissant plus rien aux programmes des usagers.

Le dernier stade apparaît avec les réseaux, et plus particulièrement avec le client serveur, car l'indéterminisme vient non seulement de ressources système nécessaires à la mise en oeuvre de mécanismes comme les RPC[43], mais également du fonctionnement du réseau sous-jacent.

La vraie question est donc: est-ce gênant? et si oui, où est la limite à ne pas dépasser?

Dans la mesure où l'information est loin des usagers, tous ces phénomènes, sauf peut-être le dernier sont indifférents à l'usager; comme cela lui simplifie considérablement son travail, c'est économiquement utile. Dans le cas de systèmes très intégrés dans le tissu socio-économique, le phénomène peut engendrer une gêne considérable, voire prendre une tournure tout à fait inquiétante dans le cas de systèmes critiques. On a alors une boucle de commande-contrôle que l'on peut représenter comme suit:



Le bruit interne *B* est:

- lié à l'indéterminisme naturel tel qu'il a été décrit ci-dessus; ce bruit est indépendant des erreurs résiduelles,
- lié aux erreurs résiduelles; il peut être amplifié par le niveau de bruit interne qui rétroagit sur lui-même.

Le paramètre historique *H* est un facteur d'«usure» lié à la durée de bon fonctionnement et à l'«histoire» des états successifs du système. Le postulat du déterminisme exige que la fonction *F* soit indépendante de *H*. Il n'en est pas de même de la durée *D* qui, bien sûr, dépend de *H*.

[43] **R**emote **P**rocedue **C**all, appel de procédure distante.

Dans le cas de réponses erronées et/ou inadaptées, l'observateur analyse la situation au moyen des fonctions inverses dont il a été question précédemment. Il est intuitivement évident que l'indéterminisme, lié au bruit, ré-injecte aléatoirement de l'information dans le processus de calcul dont il augmente l'incertitude, ce qui rend la détermination des fonctions inverses plus problématiques, voire irréalisables dans le cas où la quantité d'information contenu dans la réponse est supérieure à celle contenu dans le stimuli de départ[44]. On ne sait plus comment organiser la source froide.

Tout porte à croire que la montée de l'indéterminisme est une cause très importante de dysfonctionnement des systèmes informatiques, en particulier ceux qui font un trop large usage de la distribution des ressources; c'est d'ailleurs ce que l'on constate expérimentalement sur le terrain. Il s'agit là d'un phénomène particulièrement inquiétant sur lequel il devient véritablement urgent de se pencher, tant au plan recherche et éducation, qu'au plan industriel.

[44] Cf. Léon Brillouin, *La science et la théorie de l'information*, plus particulièrement les chapitres 19 et 20.

Quelques implications sociales en conclusion

L'histoire technologique contemporaine a vu l'émergence de phénomènes de ruptures profondes dans la relation que l'homme entretien avec sa propre production.

Rupture de l'échelle des énergies avec l'atome et les forces colossales qu'il déchaîne. Un bombe de 100 mégatonnes n'a plus rien de commensurable avec les énergies que l'on manipule ordinairement. Pour la première fois de son histoire l'homme a fabriqué un objet capable de provoquer des destructions sans commune mesure avec ce qu'il avait su faire jusque là.

Rupture de la barrière du vivant avec le décodage de l'ADN et l'émergence du génie génétique dont l'un des objectifs est de mettre l'usine cellulaire au service de la fabrication de molécules impossibles à synthétiser par la chimie traditionnelle, permettant ainsi une construction moléculaire de haute précision grâce à des fragments de programme d'ADN détournés de leur finalité première. La manipulation du vivant véhicule un potentiel d'inquiétude propre à donner le vertige aux cerveaux les mieux armés.

Rupture de la barrière spatio-temporelle qui protègeait la vie privée des individus du bruit informationnel, par le formidable coup d'accélérateur des technologies de la communication dont on sait ce qu'elles doivent à l'informatique. La bande passante de l'individu le mieux formé n'est plus en mesure de traiter de façon fiable la masse d'informations qui lui est accessible. Tout est transmis quasi instantanément et mémorisé, ce qui fait que la simple durée d'acheminement, ou la faculté d'oubli, ne sont plus en mesure de lisser et trier l'important du superflu. Il y a comme une banalisation et un aplatissement de la valeur de l'information. À quoi bon «zapper» dans une encyclopédie sur CD-ROM si l'on ne prend pas le temps nécessaire à sa compréhension. «Le temps est le père de tous les sages» disaient les présocratiques, mais cette

inflation informationnelle ne s'accompagne t-elle pas d'une perte du sens du temps, au sens physiologique du terme?

Parmi les nombreuses questions que soulève l'ingénierie de l'information, un autre nom possible du génie logiciel, j'en retiendrai deux pour conclure:

— le comportement de la machine est-il compatible avec le comportement humain?

— la machine est-elle neutre, ou risque t-elle de trahir son utilisateur?

L'homme «rationalisé»

On a vu que l'ordinateur est congénitalement déterministe; c'est sa force, mais aussi sa faiblesse dans le dialogue homme machine. Toutes les tentatives de «rationalisation» du comportement humain[45] se sont toujours terminées par des échecs — comme les tentatives de réforme des langues avec l'espéranto, le volapük,… — ou de terribles désastres — comme l'effondrement de l'URSS et le mythe de l'homme «nouveau». On pourrait en conclure qu'une certaine dose d'irrationnel est nécessaire à l'équilibre des individus et des sociétés. L'ambiguïté est après tout un moyen commode de réinterpréter ce qui a été dit et de sauver les apparences. Combien de fois chacun d'entre nous utilise t-il la formule: «ce n'est pas ce que j'ai voulu dire» très proche dans sa formulation du paradoxe d'Épiménide le crétois: «ce que je dis est faux». Toutes nos langues sont en même temps des métalangues, ce qui est bien commode, mais serait absolument fatal à tout programme construit sur ce principe!

Dans un univers rationnel, où l'ordinateur est la référence, cette capacité de réinterprétation n'est plus possible, car une vérité intangible est codée de façon indélébile dans le silicium de nos nouvelles mémoires. Il n'y a plus rien à réinterpréter.

Là est un vrai danger.

On a vu, lors du krach boursier de 89 comment l'informatique pouvait accélérer le passage d'une situation préoccupante, à une situation totalement chaotique dont l'impact économique a été considérable.

L'ordinateur n'a pas de bon sens, ni d'intuition, ni de sens des valeurs, et pas du tout de jugement, toutes choses dans lesquelles l'homme est irremplaçable[46].

A t-on par exemple analysé le potentiel de nuisance que recèle la réalité virtuelle? La réalité virtuelle ne fait pas partie de la sphère du vrai, mais comment distinguer ce qui a existé de ce qui aura été fabriqué et propagé sans frein aux millions d'abonnés Internet? C'est tout notre système de repérage visuel qui risque d'y perdre ses marques!

---

[45] Cf. G.Bateson, dans *Vers une écologie de l'esprit*, tome 2, Cinquième section, redondance et codage: «Le rêve du logicien, qui souhaite voir les hommes communiquer uniquement au moyen de signaux digitaux, dépourvus d'ambiguïtés, ne s'est pas réalisé, et ne se réalisera sans doute jamais».

[46] Le livre de R.Penrose *The emperor's new mind* rapelle quelques vérités essentielles.

Évitons donc de demander à l'ordinateur de faire ce pourquoi il n'est pas fait: l'indéterminisme. Et sachons préserver cette juste dose d'irrationnel qui rend la vie si agréable.

Le principe de neutralité

Les machines construites par l'homme sont généralement neutres vis-à-vis de leurs usagers. Elles sont cependant toutes pourvues de mécanismes de contrôles permettant l'enregistrement de paramètres nécessaires à leur entretien.

La situation change du tout au tout quand la machine manipule cette part de nous même qu'est l'information qui nous intéresse et qui appartient à cette sphère du privé. On est là face à un véritable paradoxe, car la logique de construction de la machine exige que toutes les transformations qu'elle opère soient répertoriées, ce qui rompt fondamentalement le caractère privé de l'information. Pour s'en affranchir, il faudrait mettre dans la machine une dose d'incertitude qui rendrait son utilisation douteuse et ferait perdre confiance à son usager.

Là est un autre danger.

Il faut bien comprendre que la recherche d'informations qui concernent un individu particulier, dans tous les fichiers susceptibles de contenir de l'information le concernant: banques, EDF, téléphone, sécurité sociale, péages d'autoroutes,… est une tâche de complexité linéaire pour laquelle l'ordinateur est absolument imbattable, car toutes les recherches peuvent être faites en parallèle. Le risque de dérapage vers une situation socialement inacceptable est donc parfaitement possible. Or comme disait Murphy: «if anything can go wrong, it will».

La place de l'homme

Comme l'atome, ou les biotechnologies, l'informatique n'est pas une technologie neutre. Le postulat du déterminisme qui est le fondement de son utilité véhicule un effet de bord qui, si on n'en limite pas strictement la portée, risque de transformer nos sociétés en enfer informationnel où la «vérité» de la machine finira par primer la jurisprudence humaine. Il est de notre responsabilité que l'homme ne devienne pas un simple périphérique de sa propre création.