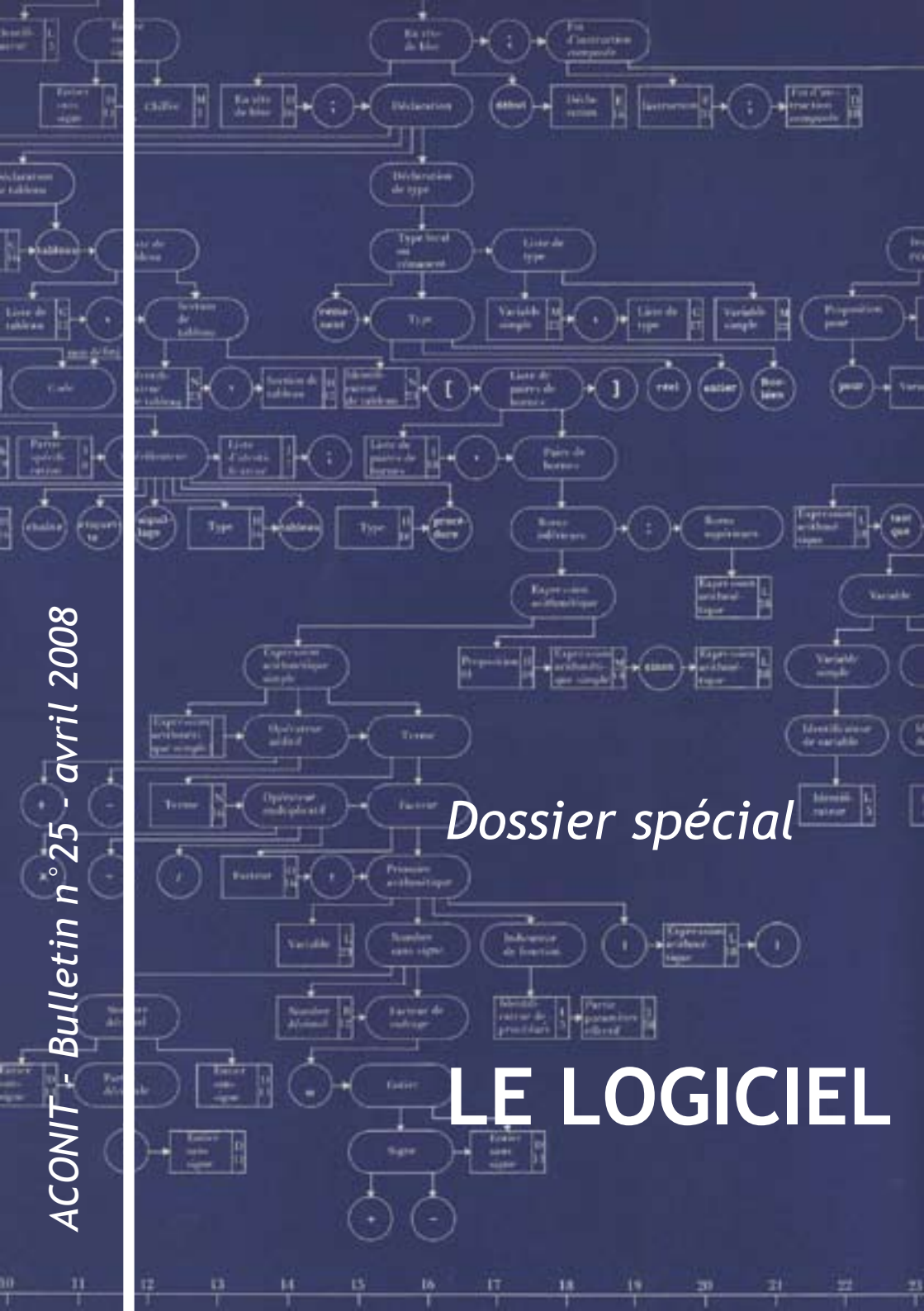


ACONIT - Bulletin n°25 - avril 2008

Dossier spécial

LE LOGICIEL

Sommaire du n° 25

Couverture : extrait de la carte syntaxique ALGOL. Dessin de Maurice Geynet. *in* L. Bolliet, N. Gastinel, P.J. Laurent, *Un nouveau langage scientifique, algol*, Paris, Hermann, 1964.

Le mot du président

Vincent Joguin

Alors que je quitte la présidence j'ai le sentiment que notre association est dans la bonne voie en direction de son objectif. L'année 2007 a été une année de renouvellement presque intégral de notre équipe permanente, et de préparation au lancement de projets ambitieux et fondateurs : réflexions sur une étude de faisabilité de notre projet, exposition sur les Interfaces Personne/Machine, prévue pour fin 2008, ouverture régulière de nos collections à la visite, projet européen de recherche et développement...

Ces projets permettront de démontrer la viabilité et l'intérêt des différentes composantes du Conservatoire de l'Informatique que nous défendons depuis si longtemps, et de permettre d'en établir la cohérence et l'originalité en regard d'autres établissements et projets centrés sur l'informatique.

La situation, au niveau national, est en effet en train d'évoluer vers une professionnalisation, et donc une marchandisation, des activités autour du patrimoine informatique. Dans ce contexte, nous devons nous positionner comme un modèle fédérateur, proposant des bonnes pratiques et un discours scientifique et technique de qualité, en pleine adéquation avec la Mission Nationale de Sauvegarde du Patrimoine Scientifique et Technique Contemporain à laquelle nous avons participé pour la troisième année consécutive.

Ainsi, en pérennisant progressivement, en partenariat avec les acteurs publics et industriels locaux, régionaux et internationaux, les différentes activités lancées dans l'année écoulée et dans les années précédentes, notre projet se mettra logiquement en place, ce qui permettra alors à plus long terme d'envisager avec ces partenaires la création d'une structure définitive encore plus professionnelle.

Enfin, nous devons remercier les collectivités locales (Ville de Grenoble, Métro, Conseil Général de l'Isère), la Région Rhône-Alpes, le Musée des Arts et Métiers, ainsi que tous nos adhérents, particuliers et organisations, pour leur soutien renouvelé.

Extrait du compte-rendu de notre Assemblée Générale du 4 avril dernier.

Mécanisme de bandes du Gamma 30

Le mot du trésorier

Pierre Thorel

À voir le résultat d'exploitation positif de l'exercice on pourrait en conclure que 2007 fut une bonne année. Il n'en est rien. Ce résultat ne fait que rattraper partiellement les déficits des années antérieures qui étaient dus à des retards de paiements séquelles de notre nouvelle installation. En fait cet exercice fut très mauvais en conséquence de notre échec à l'appel à projet de la Région Rhône-Alpes. Du fait de ce manque de ressources il nous a fallu licencier Cécile Hamadou qui assurait la médiation de nos produits et bien sûr la vie de l'association s'en est ressentie lourdement et ses recettes aussi. D'abord par l'absence de locations et de nos participations à des manifestations extérieures. Ensuite par l'arrêt momentané de notre bulletin ce qui s'est répercuté sur la diminution de nos adhérents. Cette baisse d'activité se mesure concrètement par la forte diminution de nos charges qui passent de 134 000 à 98 000 € soit une baise de 27 % !

Fort heureusement d'autres soutiens ont été fidèles. Le CNAM d'abord avec la mission de repérage et de sauvegarde du patrimoine scientifique des établissements de recherche et malgré la difficulté à mobiliser le partenariat des Universités de Grenoble. Soutien des collectivités territoriales locales dont l'aide porte surtout sur notre loyer. Nous avons hélas subit les inconvénients d'une convention triennale calculée sur le loyer de la première année et qui a laissé à notre charge les révisions annuelles des exercices suivants. Le remède à cela est que la convention est désormais annuelle et suivra au plus près l'évolution des charges. Nous avons eu aussi l'apurement de la situation avec le Conseil Général, ce qui a permis le rattrapage de notre réserve associative bien précieuse pour anticiper le fonctionnement en attente des subventions annuelles qui n'arrivent généralement pas avant l'été.

Nos adhérents de soutien Xérox et INRIA sont de solides et fidèles partenaires. Le CEA-Grenoble nous a abandonné. Mais il a été remplacé par le CNRS et l'UFR-IMA (les laboratoires de recherche en informatique grenoblois) nous rejoindra cette année.

L'année 2008 s'annonce sous de meilleurs auspices. Si Flore Gully nous a quitté en mars, nous avons pu recruter Stéphanie Lagasse en mai, dont le dynamisme et la mise au courant rapide dans des conditions difficiles nous a fait rattraper le retard d'inventaire et remplir notre contrat final.

L'agrément de notre projet auprès de la région Rhône Alpes nous permet de relancer notre activité de médiation culturelle par la conception d'une exposition sur l'interface Homme-Machine. Cette action dynamique nous oriente vers une recherche active de mécénat culturel dont nous espérons un recalibrage de notre association. Cela est rendu d'autant plus nécessaire que la région nous demande de présenter un projet de faisabilité d'un futur « conservatoire » qui sera un argumentaire détaillé et réaliste de notre développement à venir. D'autres pistes s'ébauchent dans le domaine des prestations de service et nous nous sommes entourés du personnel compétent pour cela.

C'est pour cela que le projet de budget pour 2008 est en augmentation notable notamment en ce qui concerne les frais de personnel puisque nous nous sommes en-

tourés des compétences d'une médiatrice culturelle, Constance Cazenave, depuis la fin novembre et d'un paléoinformatitien, Hans Pufal, depuis le premier janvier ; ces deux embauches se faisant avec des emplois aidés de la Région (Emploi Tremplin) et de l'État (Contrat d'Adaptation à l'Emploi). Bien entendu cela s'accompagnera de frais de structure et de frais de missions supplémentaires.

Notre association fait cette année un énorme pari sur l'avenir dans l'espoir de gagner encore en notoriété et de faire un pas décisif dans la réalisation de notre objectif d'une vitrine du passé et de l'actualité des nouvelles technologies dans le bassin grenoblois. Dans l'espoir aussi de ne pas être pris de court par des projets plus récents de musée de l'informatique en France, portés par la communication publicitaire et non par une conservation et un vécu scientifique. Leur existence exclusive détruirait beaucoup la valeur symbolique et touristique de la réalisation grenobloise d'une vitrine scientifique et industrielle.

Extrait du compte-rendu de notre Assemblée Générale du 4 avril dernier.

Hommage à Roger Gay

Michel Jacob

Je voudrais par ces quelques mots rendre hommage à Roger GAY pour le rôle essentiel qu'il a joué pour la réalisation de l'ACONIT.

C'est dans le cadre de l'ADIRA (association pour le développement de l'informatique en Rhône-Alpes) où il représentait Merlin-Gerin et moi EDF que nous avions fait connaissance.

En 1984 à la sortie d'une réunion à Marly le Roy dans les agréables bâtiments du groupe d'assurances Drouot, en attendant sur le quai le train du retour, j'ai évoqué l'intérêt d'une structure genre conservatoire pour l'informatique. Roger Gay a tout de suite approuvé cette idée qui a ensuite obtenu l'aval du groupe local de l'ADIRA auquel participait également Louis Bolliet.

C'est ensuite Roger Gay qui a obtenu le soutien de Monsieur Vaujany, PDG de Merlin-Gerin qui nous a permis de réaliser les premières embauches, une documentaliste et une secrétaire, Marie-Josée Costagliola d'Abèle (Fiat), à mi-temps!

En tant que secrétaire général pendant plusieurs années il a participé à tous les efforts pour faire avancer nos projets, en particulier auprès du maire de Montbonnot et de la municipalité de Grenoble.

Ses autres centres d'intérêt l'ont amené ensuite à se retirer de ses responsabilités à l'ACONIT.

Micro-ordinateur R2E Micral S

Que devrions-nous collecter pour sauvegarder l'histoire du logiciel ?

Len Shustek, Computer History Museum
traduction par Michel Jacob et Hans Pufal

Les historiens ont reconnu depuis longtemps la nécessité de recueillir, sauvegarder et interpréter l'histoire du logiciel pour comprendre l'histoire de l'informatique, celle du matériel étant insuffisante. Par contre, il est moins évident de savoir ce que signifie une telle opération. Je considère que si nous ne collectons, sauvegardons et interprétons les codes source en plus des objets qui en découlent, nous perdrons l'essence intellectuelle du logiciel. Mettre l'accent sur les éléments annexes focalise l'attention sur l'histoire des réalisations et sous-estime les travaux scientifiques et techniques qui les sous-tendent.

Sauvegarder le contenu intellectuel

Les musées et les archives sont les dépositaires des réalisations de nos civilisations. Pourquoi recueillent-ils et conservent-ils des objets matériels ? Les deux objectifs principaux sont leur exposition et leur étude, mais tous les objets ne permettent pas de les atteindre tous deux.

Lorsque j'accompagne les visiteurs au « Computer History Museum », la « grosse ferraille » sert d'appât pour accrocher le public. Elle évoque une époque primitive et permet de raconter les faits historiques importants. Mais les secrets que peuvent démêler les chercheurs par une étude approfondie de ces objets sont minimes et difficiles à percer. Les 15 millions de pages de documents des réserves, invisibles au visiteur occasionnel : les manuels, les études des structures logiques, les photos, l'organisation des entreprises, les documents commerciaux, l'histoire des structures juridiques ainsi que les observations des utilisateurs, voilà le contenu intellectuel des collections disponible pour une étude érudite.

Il en est de même pour le logiciel : les supports et leurs emballages sont des objets symboliques qui transmettent une information contextuelle utile, mais ne révèlent que la partie superficielle de l'histoire. Conserver le contenant du logiciel sans conserver les bits intérieurs, c'est comme conserver la couverture d'un livre sans se soucier du livre lui-même. En agissant ainsi on risque de privilégier le contenant au détriment du contenu.

De plus, conserver cartes perforées, bandes magnétiques, disques et cd-rom n'est pas conserver le logiciel. Le problème est en partie sémantique, « logiciel » pouvant désigner le support et les bits eux-mêmes. Certainement doit-on conserver les supports avec leur emballage et documentation : ils donnent un contexte utile et des indices pour la signification et l'utilisation des objets invisibles qu'ils renferment, mais ce sont les bits qui présentent le véritable intérêt. L'essence d'un livre est-elle dans le papier et l'encre, ou dans les mots ?

Programmer est plus que réaliser des produits commerciaux. C'est transformer une machine muette en un amplificateur de l'intelligence humaine. Il faut sûrement sauvegarder l'histoire des produits, mais également l'histoire du cheminement intellectuel dans la création des bits qui animent la machine.

Et dans ce but, quels bits doit-on conserver ? Le programme exécutable sous la forme du code objet ? ou le code source

écrit par le programmeur ? Les deux ont de la valeur, mais c'est le code source qui en a le plus. Le code source permet de voir à l'intérieur de l'esprit du concepteur. Nous apprenons l'utilisation prévue, les cas particuliers pris en compte, les erreurs qui se sont produites dans la conception et l'implémentation, et quels concepts ont été élaborés et utilisés. Même en « décompilant » le code objet peu de tout cela serait visible et n'est que rarement documenté. Comme le déplore Mike Mahoney en considérant les aspects les moins matériels du travail des développeurs de logiciel, « leurs idées ne sont qu'en faible partie traduites sous forme écrite ».

Le logiciel est une forme de littérature, écrite par l'homme pour être lue par des humains aussi bien que par des machines. Donald Knuth considère (défend le point de vue) que créer un logiciel est un art et dit « le but principal de mon travail d'éducateur est d'aider les gens à écrire de beaux programmes ». Qu'un programme soit traduit sous forme binaire pour être exécuté par un ordinateur ne diminue pas la valeur de ce que l'ordinateur ne voit jamais : la structure littéraire et logique, les commentaires, l'histoire de son développement dans la chronique des modifications, l'évidence des erreurs et corrections, les collaborateurs de l'auteur, l'ingéniosité, les difficultés, la fierté d'un savoir-faire, l'anxiété de la création et l'humour.

Comme le dit Andrei Ershov : « Etre un bon programmeur aujourd'hui c'est jouir du même privilège qu'un lettré du 16e siècle ». La création d'un programme beau et correct est semblable à la découverte d'une belle démonstration correcte d'un théorème mathématique.

Si nous échouons à sauver cette littérature, nous privons nos descendants d'un héritage légitime. Avoir préservé les monuments mégalithiques de Stonehenge est d'une grande valeur pour les civilisations, mais une grande part de leur objet et de leur mode de fabrication reste enveloppée de mystère.

Réciproquement, les plans de la deuxième machine de Babbage, la « Difference Engine No 2 », jamais construite du vivant de l'auteur, avaient été conservés. En disposant, 140 ans plus tard, on a pu les interpréter et construire pour la première fois cette machine. Nous avons «compilé» le code source de Babbage et compris ainsi son génie.

La voie de la conservation

Mon plaidoyer pour conserver le code source ne déprécie pas pour autant l'intérêt de la collecte des objets annexes. Au Computer History Museum notre règle, ambitieuse, pour le choix des matériaux à collecter pour un logiciel important comprend : le code source, le code objet, les versions commercialisées du programme, la documentation, les aspects commerciaux, les modes de distribution, et même les objets éphémères comme les tasses de café et les tee-shirts, les écrits et interviews, témoignages des pratiques des développeurs et des utilisateurs. C'est à partir de cette riche collection de matériaux originaux que les historiens peuvent élaborer la meilleure analyse interprétative.

Quelques supports...
Mais que contiennent-ils?

Il est vain d'espérer rassembler un tel ensemble pour de nombreux programmes, aussi le Museum a défini une double approche. Nous essayons d'être aussi complets et précis que possible pour un nombre modeste de logiciels à mettre dans le « salon », là où des objets soigneusement entretenus et documentés de façon réfléchie sont conservés selon les meilleures règles muséologiques et archivistiques.

Simultanément nous remplissons notre « grenier » d'une bien plus grande collection moins soigneusement contrôlée, dont l'origine principale sont des dons spontanés de sympathisants. Cette partie est destinée à être une énorme réserve, en majorité de codes et de manuels, mais non exclusivement. Les chercheurs peuvent l'utiliser à leur guise pour des analyses historiques mais également comme un terrain de fouilles méthodiques, d'où des objets pourront être extraits et promus plus ou moins tard vers le « salon ».

Le choix des matériaux bruts

Nous pouvons beaucoup apprendre en étudiant une large collection de logiciels représentatifs des 5 premières décades de cette activité particulière. Il n'y aura plus une autre époque où le logiciel soit embryonnaire et reste à inventer ; nous sommes passés des peintures rupestres à l'impressionnisme en une vie humaine. Il serait tragique de perdre la chronique expliquant comment ceci s'est passé.

Avec une collection arrivant à rassembler des dizaines de milliers de programmes embrassant décades, entreprises et types d'application, nous pouvons faire une analyse quantitative et qualitative et appréhender ainsi l'évolution progressive des techniques et des idées dans le développement du logiciel. Voilà le matériel brut pour l'étude de l'histoire « intime » du génie logiciel. De plus, comme les études d'antériorité pour les brevets de logiciels sont notoirement négligées, une telle banque de données est une ressource pour les recherches judiciaires d'antériorité pour les techniques de programmation. Et de toutes façons il est impossible de prévoir toutes les utilisations de telles archives. « L'utilité des données originales tient au fait qu'elles sont disponibles pour répondre à des questions qui voient le jour lors de recherches imprévisibles. »

Tout cela est possible par la seule inspection du code source. Un autre exercice entièrement différent, bien étudié par d'autres, consiste à exécuter les codes binaires sur leurs machines d'origine, soit restaurées, reconstruites ou simulées. Les difficultés de cette méthode sont réelles, mais il y a cependant aujourd'hui des centaines d'exemples de logiciels obsolètes amoureusement rendus à une nouvelle vie par la passion, l'énergie d'intelligents sauveteurs. Ce travail n'est pas à la portée de tous! Plutôt s'attaquer à une tablette d'argile babylonienne que de décoder un enregistrement de l'état d'une mémoire centrale des années 1960 (*core dump* pour les initiés!). Mais il y aura des Champollions de l'ère numérique qui étudieront, comprendront et traduiront les logiciels des premiers temps si nous sauvegardons pour eux les glyphes d'où ils pourront partir.

Ce propos a tout à fait esquivé le problème de savoir comment sauvegarder ces bits à perpétuité, une fois extraits de leurs media et documentés. La sauvegarde des logiciels historiques, une fois recueillis, est un corollaire du problème général et plus important de savoir comment sauvegarder la chronique numérique depuis le début de ce siècle inaugurant l'Âge de l'Information. Si nous n'y arrivons pas, nous serons considérés dans le futur comme responsables du premier « Âge des ténèbres numérique » et l'Histoire ne nous considérera pas avec aménité.

Le premier compilateur ALGOL sur « grosse » machine IBM, intégré au système d'exploitation IBSYS/IBJOB

J.C. Boussard,
ancien Professeur aux Universités de Grenoble et de Nice

Ce premier compilateur a été réalisé et entièrement mis au point en moins de cinq ans (1960-1964), sous la houlette aussi efficace que bienveillante des quatre patrons grenoblois de l'Informatique de cette époque – nous devrions plutôt dire de la « Computer Science » – , à savoir Louis Bolliet, pour la partie relevant essentiellement de l'ingénierie, Noël Gastinel et Bernard Vauquois – co-signataires du rapport Algol – pour les parties les plus théoriques, et enfin Jean Kuntzmann pour la supervision générale du projet dans le cadre d'une recherche conduisant à un Doctorat d'État, ces trois derniers nous ayant malheureusement quittés depuis plusieurs années.

De nombreuses autres personnes ont participé – souvent de manière déterminante – au développement harmonieux de ce projet :

• Jeunes chercheurs de la même génération que JC. Boussard lui-même : T.A. Dolotta (analyseur lexical) et Michel Berthaud (bibliothèque de procédures « standard ») pour ne citer qu'eux.

• Partenaires chevronnés de la Compagnie IBM France : François Genuys, Guy Mars et René Moreau pour l'accès commode et pratiquement illimité aux laboratoires et centres d'études de l'entreprise en programmation – entre autres ceux de La Gaude et de Corbeil-Essonnes – Alain Auroux et Maurice Guérin pour l'intégration au système, etc.

« On a fait ce qu'on a pu »

A l'époque, trois sortes de contraintes se sont révélées particulièrement difficiles à concilier avec l'ambition du projet :

• Le peu de support théorique existant du point de vue algorithmique et structures de données : théorie des « piles » de mémoires – structures *Last In First Out* – encore balbutiante (doctorat de Claude Pair à Nancy en 1965), analyse d'expressions arithmétiques simples et bien parenthésées par l'intermédiaire d'une notation « postfixée » des opérateurs (*Polish notation*), gestion améliorée des tables d'identificateurs par adressage dispersé (*Hashcoding*). Tous les autres ingrédients plus ou moins théoriques, indispensables à la bonne marche de l'ensemble, étaient à inventer de toutes pièces et à piloter de manière cohérente : se reporter au manuscrit de la thèse de JC. Boussard, comportant entre autres les « organigrammes » détaillés relevant de cette partie du travail, tous rédigés et certifiés « à la main », décrivant notamment les algorithmes d'analyse lexicale et syntaxique mis au point à cette occasion.

• Les performances encore très relatives, tant en vitesse d'exécution que de taille des mémoires principales et secondaires, des plus « grosses » machines concernées : 7090/94 et 7040/44 d'IBM, avec soumission des travaux par l'intermédiaire de cartes perforées 80 colonnes – 5000 cartes pour le compilateur définitif – et contenu de la mémoire imprimé en hexadécimal (*Core Dump*) comme unique moyen d'analyse et de « déboguage » des programmes.

Dans ces conditions, il s'est avéré particulièrement délicat – parfois inespéré... – de parvenir à des performances à l'exécution des programmes compilés pas trop éloignées – rapport optimum de 1,3 en l'absence de récursivité – de celles des langages « maison », Fortran dans la grande majorité des cas. S'agissant de l'encombrement du compilateur, rappelons que la mémoire principale de ces machines était en général adressable par mots de 6 caractères de 6 bits, pour un total allant rarement au-delà de 32 768 mots, soit appoximativement 200 *kilobytes* d'aujourd'hui, le compilateur et ses tables occupant lui-même environ 3000 de ces mots à chacun des deux « passages », soit 18 *kilobytes*!

• « *Last but not least* », le blocage administratif systématique et quasiment agressif – d'ailleurs loin encore d'être résorbé aujourd'hui, du moins en France – vis-à-vis d'une nouvelle « discipline » qui n'a pas encore de nom en 1964, au point de ne pas savoir comment libeller les doctorats en cours et à venir : le mot « Mathématiques » ne doit pas apparaître – pensez-donc! – et le mot « Ingénieur » non plus, on pourrait confondre ces travaux de vulgaire technique, menés en partenariat avec l'Industrie qui plus est, avec une recherche en la si noble Physique : impensable également! On se replie donc sur une dénomination ad hoc de « Sciences Appliquées », dont nous sommes très fiers quelques 50 ans après...

Notons malheureusement que, bien des années plus tard, le même combat d'arrière-garde, mené par de beaux esprits définitivement sclérosés, freinera pour longtemps l'enseignement et l'essor du « Génie Logiciel ». Attitude dont il semble que nous subissions encore aujourd'hui les effets ô combien dévastateurs, au simple vu de l'inadéquation lamentable et des dysfonctionnements devenus proverbiaux des grands logiciels distribués par les entreprises mondiales les plus huppées dans le domaine...

« *On a fait de notre mieux* »

En dépit des difficultés et des réticences qui viennent d'être rappelées, il est facile de constater qu'à l'époque, les équipes grenobloises – et de bien d'autres centres de recherche en Europe – ont fait de leur mieux pour parvenir dans les meilleurs délais à des produits fiables, bien intégrés au monde émergent des gros calculateurs et de leurs systèmes d'exploitation, et susceptibles d'être améliorés – voire étendus – sans difficulté par des techniciens compétents, sans crainte de voir le château de cartes s'écrouler sans préavis. A l'actif de cette recherche de qualité optimum, nous voulons citer, sans ordre de préférence préétabli :

• La complétion du langage compilé par le système de 1964 vis-à-vis de celui décrit par les rapports successifs de définition dudit langage : prise en compte intégrale de toutes les possibilités d'ALGOL 60, tant du point de vue algorithmique que structuration des données, procédures récursives à profondeur pratiquement illimitée, volume de la bibliothèque standard, précision et exhaustivité des messages d'erreurs produits tant à la compilation qu'à l'exécution, etc.

• L'intégration totale et transparente de notre système à l'ensemble IBSYS/IBJOB, permettant pour la première fois en France d'exécuter sur les machine concernées des trains successifs (*Batch Processing*) de travaux en ALGOL, mélangés sans restriction à des travaux rédigés dans les autres langages disponibles au fur et à mesure que les compilateurs adéquats sont mis sur le marché.

- L'idée originale pour l'époque – dûe à Bernard Vauquois semble-t-il – d'autoriser l'emploi dans les textes à compiler de symboles de base rédigés en français (et plus tard dans toute autre langue indo-

européenne) aussi bien qu'en anglais d'origine. Un petit « plus » bien anodin techniquement parlant, mais qui prend toute sa valeur quand on sait le temps qu'il a fallu par la suite – en raison essentiellement de la pression de multiples lobbys commerciaux et corporatistes de toutes sortes – pour amener les constructeurs à proposer des claviers comportant les signes diacritiques propres à ces langues (accents, tildes, cédilles, etc.), puis les symboles des alphabets arabes, cyrilliques, etc.

Sans parler bien entendu de l'inadéquation notoire des claviers « AZERTY » et/ou « QWERTY » à la population mondiale des praticiens de tous les jours, fort éloignée évidemment des préocupations des concepteurs d'antiques machines à écrire et des anciens personnels de secrétariat spécialement formés à grand renfort de stages accélérés pour les exploiter au mieux (les machines!)...

Rendez-vous dans vingt ans pour un clavier enfin remis dans l'ordre alphabétique et une saisie multilingue acceptée sans distinction par tout système d'exploitation digne de ce nom, chiche ?

• L'idée enfin, également très originale à ce moment-là – et redevable sans doute à Louis Bolliet en personne – de décrire les différentes phases du système dans le langage même qu'il est censé compiler et interpréter, à savoir ALGOL 60 : pan sur le bec en quelque sorte, ou recours prémonitoire aux techniques ultérieures, relevant de la méta-linguistique? Un énorme pas en tout cas, accompli dans le sens d'une meilleure compréhension des grands logiciels et de leurs échanges au sein de la communauté informatique.

Un progrès déterminant qui permet de s'affranchir définitivement de la conservation et de la communication de centaines de pages écrites en un langage de trop bas niveau pour être exploitées de manière indiscutablement fiable.

« On aurait dû mieux faire »

Avec le recul, il va de soi que les dix années écoulées après le succès des premiers systèmes que nous venons d'évoquer n'ont pas tenu, et de loin, toutes les promesses que sous-tendaient ces réussites. Les raisons de l'échec relatif (*Relative Failure*) de la lignée des langages et systèmes de type ALGOL au fil des *seventies* sont certainement à chercher dans deux grandes directions au moins :

```
DEBUT COMMENTAIRE GENERATION-METHODE DE PRECEDENCE ;
ENTIER TMAX,SYMAX,PILEMAX,REGLEMAX,STOP,AXIOME ;
.....
DEBUT
   ENTIER I,J,K,L,M,Q,LONGUEUR,NBSUJETS ;
   ENTIER TABLEAU PILE[1:PILEMAX],REGLE[1:REGLEMAX] ;
   BOOLEEN TABLEAU PE,PS[1:TMAX,1:TMAX],
                REMPLACE[TMAX+1:SYMAX,TMAX+1:SYMAX] ;
   BOOLEEN PROCEDURE TERMINAL(S) ; ENTIER S ;
      TERMINAL:=S < TMAX ;
   ENTIER PROCEDURE SYMBOLE SUIVANT ;
      DEBUT ..... FIN ;
   PROCEDURE ERREUR ; ..... ;
   PROCEDURE MACRO GENERATION ; ..... ;

.....
INITIALISATION:
   .....
   I:=J:=1 ;
   PILE[1]:=SYMBOLE SUIVANT ;
   PILE[PILEMAX]:=SYMBOLE SUIVANT ;

DELIMITATION A DROITE D UNE PHRASE PRIMAIRE:
   SI NON PS[PILE[J],PILE[PILEMAX]] ALORS
         DEBUT
            SI PILE[PILEMAX]=STOP ALORS ALLERA  FIN ;
            J:=I:=I+1 ;
            PILE[I]:=PILE[PILEMAX] ;
            PILE[PILEMAX]:=SYMBOLE SUIVANT ;
            ALLERA  DELIMITATION A DROITE D UNE PHRASE PRIMAIRE ;
         FIN ;
   LONGUEUR:= SI I=J ALORS 0 SINON I  ;
```

Extrait du compilateur ALGOL, écrit en ALGOL par Louis Bolliet dans le cadre de sa thèse « Notation et processus de traduction des langages symboliques », juin 1967.

• Le comportement plus ou moins « intégriste » d'un certain nombre d'entre nous, notamment au sein des groupes de travail de l'IFIP (WG 2.1 par exemple), chargés de trouver des successeurs crédibles à la version de 1960.

• Les intérêts commerciaux – forcément et fortement divergents – défendus d'une part par les tenants d'une programmation « scientifique » pure et dure, faisant peu de cas de l'efficacité (à tous les sens du terme) des produits finis, d'autre part par les promoteurs de techniques d'écriture et de mise au point mieux adaptées aux besoins de l'utilisateur non spécialiste, mais soucieux de performances optimales à l'exécution et d'une grande facilité d'échange de ces mêmes produits au niveau planétaire.

Sans vouloir prétendre identifier dans tous les cas et de manière fiable l'influence relative de ces deux types de raisons et de leurs conséquences, nous pensons qu'elles ont pour une large part contribué au manque d'intérêt de notre communauté pour les quatre points suivants, qui se sont évidemment révélés cruciaux par la suite :

• Simplification drastique des méthodes de définition des langages et de ces définitions elles-mêmes : cf. les succès ultérieurs de Algol W et surtout de Pascal.

• Compilation « incrémentale », vaste sujet ayant donné lieu dans les années suivantes à un grand nombre de recherches au plan mondial, avec un succès immédiat non démenti aujourd'hui.

• Modularité et compilation séparée, où ALGOL a malencontreusement laissé béante la place occupée ultérieurement par PL/1 et ADA.

• Programmation par Objets enfin, avec les développements fulgurants de EIFFEL, JAVA, etc.

Notons simplement à propos de ces deux derniers points que Noël Gastinel lui-même avait dès 1965 pressenti l'importance des réflexions à mener dans ces directions, lui qui cherchait alors à inclure à notre compilateur la possibilité de manipuler dans le langage une variable de « type » imprimante par exemple, en associant automatiquement à une telle variable toutes les propriétés qu'on pourrait en attendre : caractéristiques passives, mais aussi procédures associées, diagnostics d'erreurs appropriés, bref tous les ingrédients que ses successeurs intégreront quelques années plus tard à la notion d'Objet!

GCOS-64 : un système de gestion durable

Jean Bellec

GCOS-64 est un système d'exploitation universel conçu autour de 1970 pour supporter les ordinateurs Honeywell Series 60 level 64, une ligne d'ordinateurs moyens fabriqués par la Compagnie Honeyell-Bull dans son usine d'Angers. NEC, au Japon, en a acquis une licence et a diffusé le même système sous le nom ACOS4. En 1981, le nom de la ligne de produit fut modifié en DPS-7 (au lieu de Level-64) et GCOS64 fut rebaptisé GCOS-7. Son *kernel* a été substantiellement revu en 1986 à l'occasion de sa *release* v3B pour supporter un nouveau système de pagination. Enfin, dans les années 2000, le système a été porté sur les ordinateurs Bull Novascale 7000 à microprocesseurs Intel.

Une des caractéristiques originales de la ligne de produits GCOS fut, qu'à l'architecture relativement traditionnelle (à noter toutefois la segmentation de l'espace d'adresse servant de barrière de protection) des processeurs de la ligne, a été adjointe une couche de micrologiciel (*firmware*) standardisant plusieurs primitives importantes du système d'exploitation comme la définition et le *dispatching* des *threads* et leur synchronisation au moyen de sémaphores. Cette architecture de micro-noyau a permis une évolution sans révolution au cours de trois décennies aux besoins nouveaux de l'informatique. Elle a aussi permis une résilience particulièrement manifeste aux erreurs de programmation et même à certaines pannes de matériel.

GCOS-64, qui avait été introduit initialement pour utilisation de traitement par lots en multiprogrammation, possédait d'emblée un certain nombre de fonctions pour supporter des applications interactives et de traitements transactionnels.

Une fonction particulièrement attractive de GCOS a été une émulation efficace d'autres architectures permettant l'exécution simultanée de logiciels (système d'exploitation et programmes d'applications) destinés à ces architectures. Des émulateurs furent développés pour le GE-100, les Honeywell H-200, le Système 360 de IBM et plus tard pour les CII Siris3 et Siris8. L'objectif de performance fixe souvent l'architecture d'un émulateur : ici, tous les émulateurs partagent le concept d'un interpréteur en microcode (et par conséquent sont spécifiques du modèle émulant) et ils disposent d'un logiciel, en mode natif, pour exécuter les fonctions d'entrées-sorties et pour bénéficier de fonctions du système d'exploitation natif. Un support matériel spécifique a de plus été incorporé pour reconnaître les marques de mot sur le H-200 et l'identification des codes d'instruction pour le Siris8.

Durant les années 1990, Bull a porté le système d'exploitation UNIX sur GCOS-7 à la manière des émulateurs, sous le nom de SPIX7. Tandis que le système d'exploitation et les programmes d'applications étaient recompilés en code binaire natif, depuis leur souche écrite en C, le déroulement des applications se déroulait en parallèle avec les applications natives. Cette version d'un Unix intégré à GCOS fut poursuivie dans l'architecture Diane où Bull a développé un émulateur du code objet GCOS pour les architectures hardware Intel.

Les applications de traitement par lots (*batch processing*) étaient des travaux (*jobs*) comprenant une suite d'étapes

(*job steps*). Les travaux étaient au début lancés depuis le lecteur de cartes perforées. Des ressources (fichiers, variables de contrôle, etc.) pouvaient être passées entre *job steps* à travers le système de fichiers ou des commandes du langage de contrôle (JCL). A chaque *job step*, un groupe de processus et un nouvel espace d'adressage étaient créés. Par exemple, l'environnement du programmeur était représenté par divers groupes de processus pour l'édition du programme, la compilation et la définition des liens (*linker*).

La transition entre le traitement par lots et l'environnement interactif à partir de terminaux s'effectue en conservant la même architecture et en substituant au lecteur JCL batch, un interpréteur interactif (IOF) par utilisateur. Le langage de commande (GCL) utilisé incorporait la plupart des fonctions du JCL, mais aussi un éditeur plein écran, des menus et un dispositif d'assistance (HELP). Les terminaux étaient des écrans de type Honeywell-Bull VIP puis plus tard des ordinateurs personnels (PC) émulant les terminaux VIP. La console principale devint, elle aussi, un terminal standard, puis un ordinateur personnel.

Parce que le système IOF immobilisait un ensemble assez large de ressources pour chaque utilisateur, il ne fut pas considéré comme satisfaisant pour satisfaire les besoins d'applications transactionnelles pouvant supporter quelque milliers d'utilisateurs simultanés. Le système transactionnel diffère quelque peu des applications batch ou de mise au point interactive de programmes. Les opérateurs n'effectuent des transactions que lorsque les programmes invoqués ont été largement mis au point. De plus, le traitement des transactions implique l'introduction de nouveaux concepts de programmation comme la notion de «*commitment*», de partage dynamique de données, de journalisation, ces concepts étant absents des applications batch et de leurs langages de programmation. Bull a ainsi conçu un système de gestion de transactions – *Transaction Driven System* (TDS) – fondé sur les principes de base du système d'exploitation et du micro-noyau (primitives permanentes interprétées par *firmware*). La conception du TDS évolua progressivement pour étendre les limites placées sur le nombre d'utilisateurs simultanés pour atteindre plusieurs milliers sans en modifier les bases. Son utilisation a été facilitée par des fonctions de mise en page sur écran (FORMS) spécifiques de l'application. Plusieurs sous-systèmes TDS peuvent coexister dans un même système GCOS pour isoler des ensembles d'opérations et pour permettre un déploiement progressif des applications.

Les toutes premières versions de GCOS furent relativement lentes et représentaient un grand encombrement de mémoire par rapport aux machines auxquelles elles succédaient. En 1975, un recodage substantiel fut entrepris en particulier pour optimiser la chaîne d'opérations relatives aux entrées/sorties disques. Ceci permit à GCOS d'afficher une supériorité sur la compétition suite à plusieurs *benchmarks*.

Entre 1974 et 1979, la connexion de terminaux à GCOS se fit à travers un multiplexeur de lignes de communications connecté à l'URC (processeur de service et contrôleur d'appareils *unit-record*). Leur mise en communication par liaisons fixes ou commutées était assurée par un serveur GCOS nommé BTNS (*Basic Telecommunications Network System*). BTNS fournissait également une méthode d'accès aux applications connectées à ces terminaux (principalement IOF et TDS, mais aussi MCS - *Message control system* - application générée par les programmes COBOL).

En 1976, fut lancé le programme commun Honeywell - CII-Honeywell-Bull d'architecture de réseaux DSA (*Distributed Sys-*

tem Architecture) qui était une alternative à SNA, l'architecture de réseau IBM. Contrairement à SNA (et à BTNS) cette architecture faisait d'un processeur frontal et non plus du processeur mainframe le cœur du réseau. Le frontal DSA pouvait notamment router les communications vers différents systèmes. Le frontal choisi, nommé FNP (*Front Network Processor*) fut le mini-ordinateur Honeywell DPS-6 (nommé par CII-HB le Mini-6) doté d'un système d'exploitation spécifique du frontal et conçu à Louveciennes par les anciennes équipes de CII. Le frontal était supporté par un logiciel spécifique d'interface dans GCOS (nommé FNPS pour FNP support).

Au cours des années 1980, DSA reçut une option compatible OSI (un essai de standardisation de l'architecture par l'ISO) qui fut supportée par GCOS7 pour quelques clients.

Enfin, à partir de 1995, le triomphe de l'architecture Internet TCP/IP conduisit à une refonte de l'architecture de communications avec un frontal MainWay basé sur des microprocesseurs standard connectés par un Ethernet haut débit au central, et un logiciel Internet développé sur l'architecture Unix (soit celle de Spix7, soit celle native dans les systèmes Diane – alias Novascale 7000).

Système de fichiers (File system)

Une des spécifications initiales de GCOS-64 était de pouvoir utiliser des fichiers (le plus souvent sur bandes magnétiques), créés par les systèmes de génération précédente ainsi que sur le système IBM S/360 et d'autres systèmes étrangers. De tels fichiers n'étaient pas généralement catalogués et souvent ne portaient même pas de labels : ils devaient être gérés manuellement. GCOS-64 devait être capable de traiter ces fichiers interchangeables tout en possédant un système moderne de fichiers sur disques. Cependant, ce système de fichiers permanent dut être considéré comme une option « pour ne pas dépayser certains ateliers des clients ».

Pour faciliter le travail du client, à l'intérieur de ces contraintes, une reconnaissance automatique des volumes (sur bandes ou disques magnétiques) fut effectuée par le système dès le début de GCOS.

L'objectif de pouvoir accéder aux *disk packs* IBM avait pour conséquence de conserver le format physique des fichiers (enregistrements de longueur variable pour les fichiers séquentiels et indexés ISAM) sous le contrôle de la méthode d'accès des fichiers. Ce format fut conservé pour les fichiers natifs.

Les objets propres du système d'exploitation furent rangés et catalogués dans des bibliothèques (*libraries*) – un type particulier de fichiers dans le *file system*. Une méthode d'accès spécifique, appelée *Queued Access Method*, car elle incluait la possibilité d'augmentation de taille de la bibliothèque, gérait une allocation dynamique de blocs.

L'organisation recommandée pour les fichiers natifs était l'organisation UFAS (*Universal File Access System*) utilisée pour les fichiers séquentiels indexés et pour les fichiers d'organisation en réseau Codasyl (IDS-II). La méthode d'accès UFAS fut très améliorée sous GCOS-7 dans les années 1980 par l'addition d'un cache en mémoire (virtuelle) et par l'introduction d'un contrôle d'accès centralisé pour les accès simultanés aux bases de données à partir de *threads* parallèles (appartenant ou non à un même groupe de processus).

GCOS-7 fut aussi complété par la possibilité d'établir des miroirs de volumes, qui permettait, en outre, de partager des fichiers entre *threads* appartenant à des systèmes d'exploitation distincts (résidents dans le même système matériel ou dans des systèmes matériels différents (le

verrou temporaire étant exercé sur le fichier entier).

Les noms des fichiers catalogués avaient une longueur pouvant atteindre 44 octets, le nom pouvant être utilisé pour désigner une hiérarchie de répertoires (comme cela était dans Multics). Les privilèges d'accès pouvaient être définis grâce à ces noms.

Les fichiers catalogués pouvaient être importés et exportés sur un autre système en transférant le disque de support (les caractéristiques des fichiers étant stockés avec la table des matières du volume.

Le catalogue incluait le support des générations de fichiers soit sous forme ouverte soit en boucle (à la demande du client EDF).

Langages de Programmation

Dès le début et afin de le rendre lisible et plus facile à maintenir, il fut fait le choix d'écrire la très grande majorité du système d'exploitation en langage de haut niveau. Pour des raisons liées au choix de la « *software factory* » un dialecte de PL/1, appelé HPL (pour *Honeywell Programming Language*) fut choisi. Ce langage inclut l'accès aux principales « primitives » du système d'exploitation sous forme de macro-instructions traitées par un puissant Macro-Générateur. Le langage HPL fut au début caché aux utilisateurs afin de les décourager d'étendre GCOS de manière incompatible. Il fut finalement délivré aux grands utilisateurs et aux SSII sous le nom de GPL (*GCOS programming language*).

Le compilateur HPL fut réalisé d'abord sur notre Multics 645 (sous le nom temporaire de SHPL). Une partie du code du noyau trop dépendante du matériel a été codée en assembleur NAL pour NPL *Assembly Language* qui partageait avec HPL les mêmes définitions et types de données et les mêmes macroinstructions.

Le langage principal de programmation des clients a été, et est toujours COBOL, qui a suivi les évolutions de Cobol 68 en Cobol 74 et Cobol 60. Un pré-processeur au compilateur COBOL supporte les définitions et les verbes d'accès aux fichiers CODASYL. Le support des spécificités de programmation des applications transactionnelles a aussi requis des extensions au langage traitées par un pré-processeur. Du à l'urgence, le compilateur COBOL a fait l'objet d'un développement parallèle aux autres langages et produit directement du code binaire au lieu d'utiliser un langage intermédiaire OIL (*Optimized Intermediate Language*).

Un RPG-II (*Report Program Generator*) fut développé à partir du processeur RPG de GCOS-62 par la filiale britannique de Honeywell.

La réalisation des compilateurs pour les langages FORTRAN, C et un PL/1 standard fut en réalité plus simple à intégrer que celle de COBOL. Des composants communs à ces langages dont un générateur de code objet commun furent réalisés. Par ailleurs, le compilateur GNU du langage C fut porté sous GCOS-7 afin de faciliter le portage du sous-système UNIX.

La sortie de tous les compilateurs a été un format binaire commun (appelé *compile-unit format*) permettant des appels de procédures écrites dans différents langages. Un éditeur de Liens (*static Linker*) permet d'allouer des segments aux *compile-units*, de construire les blocs de contrôle correspondant aux espaces d'adresses et de créer des objets du système d'exploitation relatifs au programme (tels que les blocs de contrôle des *threads* et les sémaphores). Un des rôles principaux du *linker* est de résoudre les références entre le programme utilisateur et les bibliothèques du système. Cependant, la résolution définitive de ces liens est faite au moment du chargement du programme en mémoire virtuelle par un programme

appelé *Loader* qui alloue les numéros définitifs de segments aux objets partagés avec le système.

Il est ainsi possible de réaliser la préparation des programmes (Macrogénération, Compilateur, Editeur de Liens) sous un système différent de celui de production.

Le choix d'un mot de 32-bits pour l'architecture du système avait l'inconvénient de provoquer des restrictions au nombre de segments disponibles pour un *thread*. Si l'option par défaut resta une assimilation d'une procédure externe à un segment code, cela devenait prohibitif au moment de la résolution des références par exemple à la bibliothèque de fonctions scientifiques. Pour contourner ce problème, un *Binder* (relieur) était capable de fusionner plusieurs *compile-units* en une seule en utilisant la facilité de l'architecture associant plusieurs points d'entrée à un segment procédure.

Le nombre de grands segments (4 Mo) allouable au moment de l'édition de liens était encore plus limité que celui des petits segments de 64 Ko, si bien que GCOS a créé un mécanisme de pagination des grands ensembles de données (par exemple un tableau Fortran ou la pile de PL/1 ou de C) en les pageant automatiquement sur un petit nombre de petits segments.

Outre les langages précédents, à la fin des années 1970, le besoin se fit sentir d'offrir les langages BASIC et APL (« A Programming Language » de Iverson) dans un environnement interactif. Dans la seconde moitié des années 1990, ce fut le langage JAVA qui sera implémenté. Contrairement aux langages (à dominante batch) ces langages furent interprétés par un environnement spécifique (ex : JVM *Java Virtual Machine*) tandis que le langage source était transformé en une séquence de caractères par un pré-processeur interactif.

D'autres processeurs assimilables aux processeurs de langages furent aussi réalisés dans l'environnement IOF, ce furent un éditeur de texte en mode ligne (utilisé pour l'entrée de programmes) et un traitement de texte (Wordpro) répondant à l'état de l'art des années 1970.

Bases de Données

Le premier système de base de Données, développé depuis le début (en liaison avec le système transactionnel TDS en 1977) a été IDS-II (*Integrated Data Base System*), en conformité avec les spécifications CODASYL. La première version comprenait seulement une vue conforme au schéma de la base de données. Les multiples vues (subschemas) furent implémentées plus tard à la fin des années 1980. IDS-II sur GCOS-7 permettait aussi à un programme de supporter simultanément plusieurs bases de données IDS-II ainsi que des fichiers UFAS. Un processeur interactif de langage d'interrogation de bases de données IQS (*Interactive Query System*) permet aussi de consulter des bases simultanément avec une utilisation transactionnelle sous TDS.

A côté de IDS, un autre système moins ambitieux, conçu sur le GE-58 et développé sous GCOS-62, MLDS, fut réalisé pour aider à la migration des clients.

En 1982, Bull fit le choix de porter le système de bases de données relationnelles ORACLE sur GCOS-7. Oracle n'était alors qu'une petite start-up californienne développant son système pour AVX, UNIX et IBM. Bull effectua sous licence le transcodage (utilisant le langage C). Bull développa aussi des utilitaires d'importation et d'exportation de données d'Oracle vers UFAS et IDS-II. Conformément au *design* d'Oracle Corp, la réalisation se fit sous forme d'un serveur spécifique dans GCOS. Un Dictionnaire de Données capable de gérer les fichiers et les entités bases de données fut développé en 1981.

Dialogue avec les ordinateurs personnels (Micro-Mainframe links)

Dès le début des années 1980, les micro-ordinateurs furent capables d'être utilisés dans un environnement GCOS en utilisant leurs émulateurs de terminaux et en détournant vers leurs fichiers privés la sortie des messages. La communication avec des ordinateurs personnels ou avec les stations Questar 400 fut réalisée en 1986 en regroupant le support d'accès aux fichiers UFAS et IDS-II (via IQS) dans un *pool* d'agents dans le programme AFFINITY.

De plus, un utilitaire de transfert de fichiers TEMPUS-LINK fut réalisé sous licence de MicroTempus.

Bureautique

Vers 1985, Bull a commercialisé DOAS7 pour la ligne GCOS-7. DOAS7 était un nom générique pour des applications de bureautique communicantes. Il combinait DFA7 (un système d'archivage de documents), un système de courrier X400, le système d'indexation de documents MISTRAL, chacun ayant son propre agent utilisateur sur PC ou sur Questar 400. DFA7 (*Document Filing Application*) fut initialement développé par la SSII Méthodes et Informatique, et fut ensuite acquis par Bull. DFA7 fut développé en COBOL/IDSII sous forme d'un produit transactionnel TDS. Par contre MISTRAL, un produit déjà développé par CII avant 1975, fut codé en Fortran et s'exécutait interactivement sous IOF.

En réalité, ces produits bureautiques sur « mainframe » ont été rapidement concurrencés par la combinaison micro-ordinateurs - serveurs de fichiers et seul MISTRAL survit encore dans les années 2000.

Applications

Initialement, la stratégie pour GCOS-7 avait été d'offrir un catalogue d'applications universelles développées exclusivement au sein de la compagnie par le groupe Applications. Après 1978, cette stratégie s'ouvrit aux développeurs extérieurs afin d'ouvrir la base d'applications et espérer concurrencer IBM.

Le manque d'expérience chez Bull dans le domaine de la planification automatique des travaux fut surmonté par des développements faits par les SSII Unilog, Genlog... qui développèrent des générateurs de GCL incluant des utilitaires de comptabilité et de planification.

Le port de UNIX sous GCOS-7 a permis l'exploitation d'applications développées pour UNIX sous forme de serveurs spécialisés accessibles de l'environnement GCOS. En particulier, c'est dans le serveur UNIX sous GCOS que fut initialement réalisé les programmes d'interface entre les applications GCOS et le réseau Internet.

Conclusion

Aujourd'hui, le système reste encore utilisé sur les systèmes matériels DPS-7000. Le support de la fonction de partitionnement en plusieurs copies de GCOS a permis une grande flexibilité dans l'opération de ces machines.

D'autre part, le Novascale 7000 est une machine multiprocesseurs à base Intel et est capable d'exécuter sous une même configuration Linux, GCOS-7 et éventuellement Windows. L'architecture matérielle du DPS-7000 est émulée sur les processeurs Intel et la totalité de GCOS et de ses applications, en particulier celles sous TDS, s'exécute inchangée dans cet environnement, à l'exclusion des entrées-sorties physiques.

Sur l'utilisation dangereuse des instructions « aller à »

Go To Statement Considered Harmful (mars 1968)

Edsger W. Dijkstra,
Université technologique, Eindhoven, Pays-Bas
traduction par Louis Bolliet - 2008

Depuis des années je me suis rendu compte que la qualité des programmeurs est une fonction décroissante du nombre des instructions « aller à » qu'ils utilisent dans l'écriture de leurs programmes. Plus récemment j'ai trouvé pourquoi l'usage des instructions « aller à » avait des effets désastreux, et je me suis convaincu que les instructions « aller à » devraient être abolies de tous les langages de programmation évolués (tous sauf le langage machine). A cette époque je n'ai pas attaché beaucoup d'importance à cette découverte ; si je propose aujourd'hui de publier mes arguments c'est en raison d'un fort encouragement à le faire suite à des discussions récentes sur ce sujet.

Ma première remarque : bien que l'activité du programmeur soit terminée quand il a construit un programme correct, le processus qui s'exécute sous le contrôle de son programme est le point important de son activité car c'est ce processus qui doit réaliser l'effet désiré ; c'est ce processus qui dans son comportement dynamique doit réaliser les spécifications souhaitées. Une fois que le programme est écrit, la réalisation du processus associé est déléguée à la machine.

Ma seconde remarque est que nos capacités intellectuelles sont assez aptes à maîtriser les relations statiques mais mal adaptées à se représenter des processus dynamiques dans le temps. Pour cette raison nous devons tout faire (comme programmeurs conscients de nos limitations) pour diminuer le décalage conceptuel entre le programme de nature statique et le processus de nature dynamique, pour que la correspondance entre le programme (étalé dans le texte) et le processus (étalé dans le temps) soit aussi simple que possible.

Considérons maintenant comment nous pouvons caractériser la progression d'un processus. (Vous pouvez réfléchir à cette question d'une manière très simple : supposons qu'un processus, considéré comme une succession d'actions dans le temps, soit arrêté après une action arbitraire, quelles données devons-nous enregistrer afin de pouvoir ré-éxecuter le processus jusqu'à cette même action ?).

Si le programme est une simple suite d'instructions d'affectation (pour cette discussion considérées comme les descriptions d'actions élémentaires) il suffit d'un pointeur dans le texte du programme entre les descriptions de deux actions successives.

(En l'absence d'instructions « aller à » je peux me permettre l'ambiguité syntaxique dans les trois derniers mots de la phrase précédente : si nous les analysons comme « succession de descriptions d'actions » nous signifions succession dans l'espace du texte ; si nous analysons comme « description d'actions successives » nous signifions succession dans le temps).

Définissons par index textuel un pointeur à un certain endroit du texte.

Quand nous incluons les instructions conditionnelles (*if B then A*), les instructions alternatives (*if B then A1 else A2*), les instructions de choix telles qu'introduites par C.A.R.Hoare (*case<i> of (A1, A2, ..., An)*) ou les expressions conditionnelles introduites par J.McCarthy (*B1_E1, B2_E2, ..., Bn_En*) la progression du processus est toujours définie par un seul index textuel.

Dés que nous incluons les procédures dans notre langage nous sommes forcés d'admettre qu'un seul index textuel n'est plus suffisant. Dans le cas ou un index textuel est utilisé à l'intérieur d'une procédure la progression ne peut être repérée que si nous précisons à quel appel de la procédure nous faisons référence. Avec l'inclusion des procédures nous pouvons caractériser la progression du processus par une suite d'index textuels, la longueur de cette suite étant égale à l'imbrication dynamique des appels de procédures.

Considérons maintenant les instructions de répétition (*while B, repeat A* ou *repeat A until B*). Logiquement ces instructions sont maintenant inutiles car nous pouvons exprimer les répétitions avec des procédures récursives. Pour des raisons pratiques je ne souhaite pas les exclure : car d'une part les instructions de répétition peuvent être facilement réalisées sur les machines actuelles ; d'autre part le schéma de raisonnement par induction nous permet de bien comprendre les processus générés par des instructions de répétition.

Avec l'inclusion des instructions de répétition, les index textuels ne permettent plus de décrire la progression dynamique du processus.

Avec chaque entrée dans une instruction de répétition nous pouvons associer « un index dynamique » qui compte en permanence le rang de la répétition en cours.

Etant donné que les instructions de répétition, (de même que les appels de procédure) peuvent être imbriquées, nous constatons que la progression du processus peut toujours être caractérisée de manière unique par une suite (mixte) d'index textuels et/ou dynamiques.

Le point essentiel est que les valeurs de ces index ne sont pas contrôlées par le programmeur ; ils sont générés – qu'il le souhaite ou non – par l'écriture du programme ou par l'évolution dynamique du processus.

Pourquoi avons-nous besoin de ces coordonnées indépendantes ? La raison est que nous pouvons interpréter la valeur d'une variable uniquement en fonction de la progression du processus (ceci est inhérent aux processus séquentiels). Si nous voulons compter le nombre n de personnes dans une salle initialement vide, nous pouvons réaliser ce comptage en augmentant n de 1 chaque fois que nous voyons quelqu'un entrer dans la salle. Dans l'intervalle de temps entre l'instant où nous voyons quelqu'un entrer dans la salle et l'instant où nous augmentons la valeur de n, la valeur de n est égale au nombre de

personnes dans la salle moins 1. L'utilisation sans contrainte de l'instruction « aller à » a comme conséquence immédiate qu'il devient très difficile de trouver un système de coordonnées adéquat pour décrire la progression du processus. Habituellement, on prend en compte les valeurs de quelques variables bien choisies mais c'est inadéquat car c'est en relation avec la progression que la signification de ces valeurs doit être comprise.

Avec l'instruction « aller à » on peut toujours décrire la progression avec un compteur qui compte le nombre d'actions réalisées depuis le début du programme (une sorte d'horloge normalisée). La difficulté est qu'une telle coordonnée, bien qu'elle soit unique, ne sert strictement à rien. Dans ce système de coordonnées il devient extrêmement compliqué de définir tous les points de la progression où n est égal au nombre de personnes dans la salle moins 1!

L'instruction « aller à » est trop primaire : c'est essentiellement un encouragement à écrire des programmes confus. On peut considérer l'utilisation des instructions conditionnelles, de choix, de répétition comme une limitation.

Je ne prétends pas que ces instructions puissent satisfaire tous les besoins mais que quelles que soient les formes d'instructions proposées (comme par exemple les instructions d'alarme erreur) elles doivent permettre qu'un système de coordonnées indépendantes du programmeur puisse décrire simplement et utilement le processus.

Dois-je reconnaître qui m'a influencé dans mes réflexions ? Il est évident que ce n'est pas sans l'influence de Peter Landin et Christopher Strachey.

Finalement je dois mentionner (car je m'en souviens très bien) comment Heinz Zemanek au meeting sur la version préliminaire d'ALGOL en 1959 à Copenhagen a exprimé de manière très explicite ses doutes pour placer au même niveau syntaxique l'instruction « aller à » et l'instruction d'affectation.

Je m'excuse très simplement de ne pas avoir tiré les conséquences de sa remarque dès cette époque.

La remarque concernant l'inconvénient de l'instruction « aller à » n'est pas récente. Je me souviens d'avoir lu la recommandation explicite de restreindre l'utilisation de l'instruction « aller à » pour les alarmes erreur mais je n'ai pas pu en retrouver la trace ; je présume qu'elle est due à C.A.R.Hoare. Dans l'article « 1,Sec. 3.2.1 », Wirth and Hoare font la même remarque dans la présentation de l'instruction de choix :

« Comme l'instruction conditionnelle, elle fait apparaître la structure dynamique d'un programme de manière plus claire que les instructions "aller à" et les instructions d'aiguillage et elle supprime l'introduction d'un grand nombre d'étiquettes dans le programme. »

Dans l'article « 2 », Guiseppe Jacopini a semble-t-il démontré que l'instruction « aller à » n'est pas indispensable. Mais il n'est pas recommandé de traduire un organigramme arbitraire de façon mécanique en un organigramme sans « aller à », car l'organigramme résultant ne sera pas plus lisible que l'original.

Références :

1. Wirth, Niklaus, and Hoare C.A.R. A contribution to the development of ALGOL. Comm.ACM 9 (June 1966) , 413-432.

2. Böhm, Corrado, and Jacopini Guiseppe. Flows diagrams, Turing machines and languages with only two formation rules. Comm ACM 9 (May 1966), 366-371.

Des nouvelles de l'atelier mécanographie

Hans Pufal et Constance Cazenave

Le 24 mai prochain, le Collège de l'Europe de Bourg de Péage fête son centenaire. À cette occasion, le professeur de technologie de l'établissement, qui connaissait l'association grâce à une présentation de notre mallette pédagogique, nous a contactés afin de l'aider à réaliser une petite exposition sur l'histoire de l'informatique. L'une de nos expositions sera donc présente lors de cette journée, ainsi que quelques machines emblématiques de l'informatique dans les collèges, TO7, MO5... tous en fonctionnement.

Cette exposition présentera également une perforatrice de cartes, objet probablement inconnu des collégiens d'aujourd'hui. Cela a été pour nous l'occasion de remettre au goût du jour l'atelier mécanographie...

Le succès de l'atelier mécanographie d'ACONIT...

Au début des années 2000, Jacques Pain a remis en route une trieuse. Alors est née cette idée d'un atelier mécanographie au sein de l'équipe ACONIT. Agnès Félard et Muriel Battistella, alors employées de l'association, ont décidé de faire participer ACONIT au Prix Carrefour ; les délais de participation étaient pourtant assez courts. Et ACONIT, pour son projet de conservatoire et cet atelier mécanographie, a remporté le Premier Prix du Patrimoine Culturel! Cet atelier, dont la trieuse à nouveau en état de marche était la pièce maîtresse, permettait notamment aux visiteurs de découvrir le codage des cartes perforées et de perforer eux-mêmes des cartes qui étaient ensuite triées.

Fort de ce succès, l'atelier continue sa route... Après la Fête de la Science, il est exposé lors du 90e anniversaire d'IBM France. Dans ce cadre, notre trieuse voyage notamment jusqu'à Rolland Garros mais aussi au siège d'IBM à Paris où les employés d'IBM ont ainsi pu (re)découvrir l'histoire de leur entreprise.

Par la suite, l'atelier mécanographie revient dans la région : il rencontre un très grand succès dans les superbes salles de l'Hôtel de Ville de Lyon lors des Journées du Patrimoine, à la Halle Tony Garnier, mais aussi, plus insolite, dans la salle d'exposition de la FNAC de Grenoble.

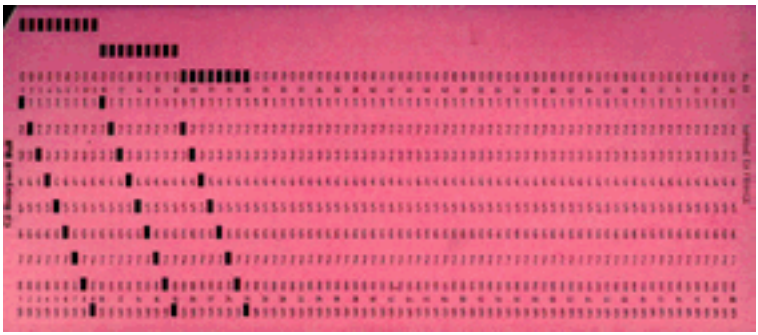
2008 : l'atelier mécanographie à nouveau sur la route...

Le 24 mai, la trieuse ne fait malheureusement pas partie du voyage pour des raisons pratiques.

Chaque élève qui le souhaite pourra cependant perforer son prénom et son nom sur une carte qu'il pourra garder en souvenir de cette journée. Nous vous présentons ici la nouvelle « fiche exemple » à destination des élèves. Cette fiche sera présentée à côté de la perforatrice et chaque élève recevra également une fiche sur laquelle il pourra coder ses nom et prénom. Tous vos commentaires sont évidemment les bienvenus pour améliorer encore cet atelier qui rencontre toujours autant de succès.

Nous ne manquerons pas de publier quelques photos de cette journée sur notre site web.

CODAGE D'UNE CARTE PERFORÉE



Exemple : l'alphabet sur une carte perforée

Observe cette carte perforée : chaque lettre de l'alphabet est codée dans une colonne grâce à deux trous. La position de ces trous correspond aux coordonnées que tu trouves dans le tableau ci-dessous.

A : 1 - 12 B : 2 - 12 C : 3 - 12 D : 4 - 12 E : 5 -12	F : 6 - 12 G : 7 - 12 H : 8 - 12 I : 9 - 12 J : 1 -11	K : 2 - 11 L : 3 - 11 M : 4 - 11 N : 5 - 11	O : 6 - 11 P : 7 - 11 Q : 8 - 11 R : 9 - 11	S : 2 - 0 T : 3 - 0 U : 4 - 0 V : 5 - 0	W : 6 - 0 X : 7 - 0 Y : 8 - 0 Z : 9 - 0

Codage de l'alphabet sur cartes perforées

Maintenant, tu vas apprendre à coder ton prénom et ton nom pour ensuite perforer la carte.

Aide-toi de cet exemple :

a	*c*	*o*	*n*	*i*	*t*					

trou n°1	*1*	*3*	*6*	*5*	*9*	*3*					
trou n°2	*12*	*12*	*11*	*11*	*12*	*0*					

Il est temps de perforer ta carte. Introduis une carte dans la perforatrice. Pour perforer une lettre, tu dois appuyer **en même temps** sur les deux touches de ses coordonnées. Par exemple, pour faire un A, appuie en même temps sur les touches (1) et (12).

Entre ton prénom et ton nom, appuie sur la touche (E) pour les espacer.
Quand tu as fini, appuie sur la touche rouge pour récupérer ta carte.

Félicitations, tu as perforé une carte avec ton prénom et ton nom!

Association pour un conservatoire de
l'informatique et de la télématique
12 rue Joseph Rey - 38000 Grenoble
Tél. +33 (0)4 76 48 43 60
Mél. info@aconit.org
Web www.aconit.org

L'association pour un conservatoire de l'informatique et de la télématique (ACONIT) a été créée en 1985, à Grenoble, par des ingénieurs d'EDF et Merlin-Gérin avec le parrainage de personnalités de l'Université et de l'industrie. ACONIT a reçu le soutien de nombreuses institutions et organismes nationaux, régionaux et locaux.
Les missions de l'ACONIT sont :

- conserver du patrimoine matériel, intellectuel et des savoir-faire constitués au cours de l'évolution de l'informatique. Mettre ce patrimoine à la disposition de tous ;
- contribuer au développement et à la diffusion de la culture scientifique et technique auprès du grand public ;
- susciter et soutenir des recherches pluridisciplinaires pour mieux comprendre l'informatique et ses interactions avec la société.

ACONIT a constitué une des plus importantes collections européennes de matériels, de logiciels et documentations techniques et commerciales illustrant l'histoire de l'informatique.

Recevoir une copie électronique du bulletin ?

Si vous le souhaitez, nous pouvons vous envoyer notre bulletin sous forme électronique (fichier PDF).

Si cette diffusion « express » vous intéresse, merci de le signaler à Constance (ccazenave@aconit.org), et l'association fera quelques économies de timbres…