Objets inanimés, avez-vous un langage ?

PAR FRÉDÉRIC MAURE

Est-ce le mouvement des bateaux dans le fjord d'Oslo, nul ne sait pourquoi des universitaires norvégiens adeptes d'Algol 60 ont eu l'idée de rompre avec la programmation algorithmique classique...

Le langage qu'ils ont créé en 1967, Simula, permet de décrire un modèle général de bateau, un modèle de grue, un modèle de dock, chacun avec ses capacités et ses actions potentielles, indépendamment du programme principal qui appellera le bateau Marie-Galante, la grue G12, le dock Est, objets construits sur leur modèle.

FJORD : PERTURBATIONS DANS LES OBJETS

La simulation de processus qui interagissent - d'où *Simula* tire son nom - est en effet très difficile à programmer en *Fortran* ou en *Cobol* parce qu'elle oblige à décrire à l'avance toutes les interactions possibles entre tous les états des variables du problème. La cascade d'alternatives et leur combinatoire n'est pas maîtrisable. Des modèles d'objets arrêtés une fois pour toutes permettent par contre de décrire un état précis du problème sous la forme d'un ensemble d'objets. Il suffit ensuite de lancer une perturbation (un changement d'un objet, la création d'un nouvel objet, etc...) pour qu'elle se propage d'objet en objet, en suivant les réactions de chacun, conformément à son modèle.

Il y a ainsi beaucoup de problèmes dont il est trop long et trop coûteux d'essayer de décrire de manière fiable et complète l'algorithme général de résolution. Celui-ci reste le squelette du programme écrit dans les bons vieux langages de programmation séquentiels. Les objets de Simula contiennent des algorithmes partiels qui leur sont propres et qui se déclenchent quand le programme principal le demande à l'objet. La complexité de la programmation est alors réduite.

LACS SUISSES : LES TYPES...

A peu près à la même époque, les lacs suisses inspiraient à l'universitaire N. Wirth le langage *Pascal*, qui offre au programmeur la capacité d'exprimer une structure de données complexe avec des fonctions d'accès : un type. Le but des types est de contrôler les appels de fonctions pour ne pas essayer d'additionner des choux et des carottes. Mais les types sont aussi une manière de décrire un problème à un plus grand niveau d'abstraction que les variables de base du langage, comme dans les années pionnières de l'informatique où règnait la codification et où Mercredi n'existait que comme "3". Du pont de vue de la conception de programme, les types ressemblent beaucoup aux objet des Norvégiens. Pour le programmeur, le résultat est une plus grande sécurité et une meilleure concision du code.

BAIE DU MAINE : LES ENTITÉS...

Du côté des lacs du Vermont, ou bien était-ce une baie du Maine ?, un certain M. Codd (sic...) réfléchit à la même époque à un formalisme adapté à la conception d'applications informatiques, bien en amont de la programmation. Il propose de représenter chaque notion centrale d'une application par une entité regroupant l'ensemble des informations élémentaires qui la caractérisent. Chaque entité d'informations est indépendante des autres entités. Les informations composées sont représentées par des relations entre entités : une Facture est une relation entre un Client et un Modèle de voiture. Les traitements opèrent ensuite sur les occurrences d'entités et de relations, mais chaque occurrence est conçue sur le modèle de son entité ou de sa relation : la voiture bleue immatriculée 4421 HVA 75 est une occurrence de Renault 25.

En informatique de gestion, le modèle Entité-Relation permet de définir des structures de fichiers ou de base de données. Chaque enregistrement du fichier correspond à une occurrence. C'est un moyen de hiérarchiser les données d'une application. Il repose sur l'hypothèse que les données sont la partie la plus stable d'un logiciel, et que les erreurs de conception sur les structures de données sont beaucoup plus coûteuses à réparer que les bogues dans les traitements.

Objets, types et entités sont nés dans des laboratoires universitaires et le monde *Cobol-CICS* les a longtemps boudés. Ce n'est plus le cas : vingt ans après, ils font partie de l'informatique professionnelle. Aucune méthode de conception d'application n'ignore le modèle Entité-Relation : c'est par exemple une des bases de *Merise*. Les types sont devenus si précieux pour la correction des programmes que *Pascal* lui-même est utilisé dans les entreprises. *Le langage* C per-

met le typage et *Ada*, créé en 1980 à la demande du Département de la Défense américain comme son aîné *Cobol*, contrôle la cohérence des types avant même d'examiner le corps des procédures. Les langages à objets, perfectionnés et diversifiés depuis *Simula*, commencent eux aussi à être utilisés pour nombre d'applications conséquentes. L'une des plus célèbres est un système d'exploitation du Macintosh, réalisé en 1984 à l'aide d'une extension objets de *Pascal*.

La programmation par objets

Il existe maintenant de nombreux langages informatiques qui permettent de déclarer des structures de données et des procédures associées. Ici, l'important est "déclarer". Non pas que la programmation des procédures propres à l'objet soit déclarative au sens de *Prolog* ; au contraire, les langages à objets utilisent dans leur grande majorité des structures de contrôle semblables à celles des langages classiques, voire les reprennent à l'identique (c'est bien sûr le cas des extensions objets des langages impératifs existants : *Pascal*, C, etc.).

Déclarer un objet, c'est le programmer indépendamment des autres objets. La clef de la déclaration d'un objet est le partage entre les informations publiques, accessibles par l'extérieur de l'objet, et les informations privées qui décrivent la structure interne et auxquelles il n'est pas possible de toucher directement : il faut passer par les fonctions d'accès publiques (déclarées comme telles). Quant le programmeur demande par exemple à un bateau-objet d'avancer (nom public de la procédure), c'est le corps de la procédure (privé) qui vérifie si le bateau a du carburant.

La résolution d'un problème est disséminée dans les algorithmes partiels des comportements des objets, comme dans le cas de la simulation de processus : un bateau peut avancer de x mètres, mais pas un dock ; une grue peut décharger un bateau, mais un bateau ne le peut pas. Ce sont des lois de comportements potentiels, qui seront ou non activées lors de l'exécution du programme, mais qui sont déclarées chacune indépendamment des autres et non dans une cascade d'alternatives.

Classes et tendances : l'age du capitaine...

La définition ci-dessus est d'ailleurs un peu impropre : on ne programme pas directement des objets mais leur modèle, ou *classe* dans la terminologie classique. Un objet est l'*instance* d'une classe, il est fabriqué sur le modèle déclaré. Cela rappelle les occurrences d'entités du modèle Entité-Relation, mais il ne s'agit dans ce cas que de données élémentaires, alors que les objets contiennent des données structurées et des procédures. Avec un langage à objets, on programme des classes et le langage exécute des instructions sur des instances : la Marie-Galante est longue de 48 m, elle est capable d'avancer sur l'eau de x mètres, mais elle ne peut pas décharger de bateau, conformément à son modèle. C'est une instance de la classe Bateau caractérisée par les variables Longueur, Port d'attache, Localisation, Capitaine et les comportements Avancer et Etre au mouillage.

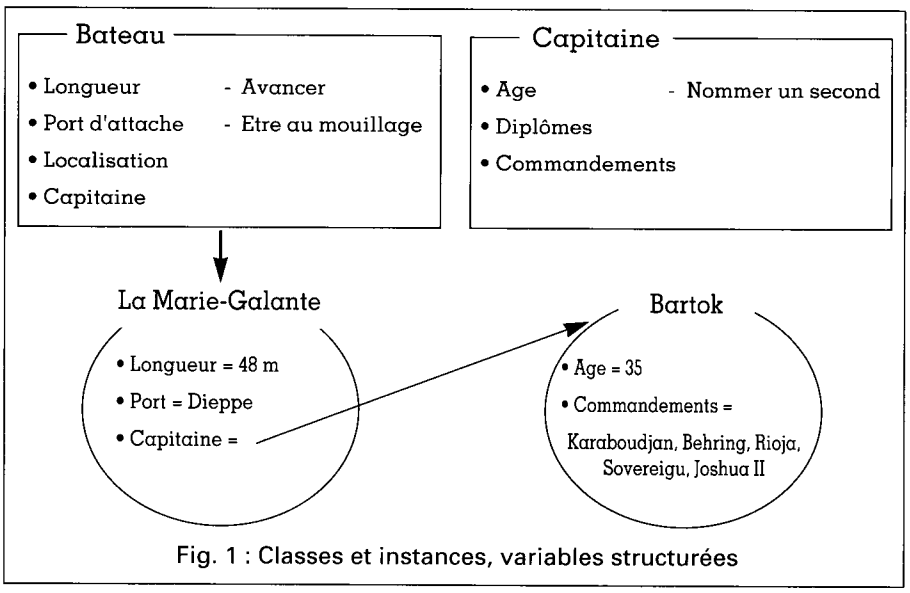

Fig. 1 : Classes et instances, variables structurées

Un langage à objets permet d'attribuer aux variables des valeurs plus subtiles que 48 m. La capitaine Bartok qui commande la Marie-Galante peut tout-à-fait être lui-même une instance de la classe Capitaine caractérisée par les variables Age, Diplômes, Commandements et le comportement Nommer un second. Les variables ne sont pas plates mais structurées conformément à leur type-classe. Il est ainsi possible de programmer directement l'Age du Capitaine du Bateau.

Heritages entre classes

Déclarer des classes demande de penser à un certain niveau d'abstraction par rapport aux objets concrets qui seront manipulés lors de l'exécution du programme. Mais il y a d'autres niveaux d'abstraction. A partir de la classe Bateau, il est possible de déclarer une *sous-classe*, par exemple Bateau à moteur. Elle *hérite* des caractéristiques de la classe Bateau, pas la peine de les répéter. Mais elle contient en plus les variables Puissance et Quantité de carburant et le comportement Ravitailler. Naturellement, une sous-classe de Bateau à moteur peut-être la classe Pétrolier, etc.

Avec cette structure de classes et sous-classes, la Marie-Galante est maintenant une instance indirecte de Bateau. Les langages à objets utilisent l'héritage entre classes pour étendre la modélisation d'un problème en spécialisant une classe par une sous-classe ou en factorisant dans une sur-classe des informations communes à deux classes (Bateau est sur-classe de Bateau à moteur et Voilier).

L'héritage entre classes donne beaucoup d'espoir à ceux qui recherchent l'extensibilité des logiciels, voire leur réutilisabilité dans de nouveaux contextes. Plus modestement, l'héritage facilite la mise au point incrémentale et surtout offre la possibilité de programmer directement les exceptions de l'analyse (à ne