Apport de l'Orientation-Objet dans le développement d'applications informatiques

F. Becker

Le développement d'applications informatiques plus ou moins importantes fait partie des activités de nombreux professionnels de la topographie. Parmi les méthodologies possibles pour ces développements, l'orientation-objet tient une place de choix comme en témoigne le succès de langages comme C++ ou Java qui permettent sa mise en œuvre. Cet article vise à illustrer les bénéfices d'une telle approche.
Nous nous fonderons sur l'expérience que nous avons acquise lors du développement de la nouvelle version du logiciel de compensation employé pour les calculs de métrologie des accélérateurs du CERN, aussi nous illustrerons notre propos avec des exemples issus de ce programme.

Contexte

Des projets de métrologie de l'ampleur de ceux qui sont conduits au Laboratoire Européen pour la Physique des Particules (CERN) nécessitent un outil performant de calcul de compensation par les moindres carrés dans un référentiel 3D local.

Le logiciel LGC (Logiciel Général de Compensation), initialement développé par Michel Mayoud et utilisé depuis près de vingt ans dans le groupe de métrologie du CERN, répond à ce genre de besoins. Son code source représente environ 10 000 lignes de Fortran. Il présente l'inconvénient majeur d'avoir subi de nombreuses mises à jour si bien que sa maintenance est devenue très

hasardeuse. De plus des améliorations devaient être apportées, et c'est ce qui fut entrepris par le groupe sous la conduite de Mark Jones. Enfin, l'incorporation nécessaire de nouveaux types d'observations dans le cadre du projet CLIC (Compact LInear Collider) **[2]** conduisit à développer rapidement une version spéciale du logiciel.

Celui-ci doit désormais présenter une grande souplesse de mise à jour afin qu'il puisse rapidement être adapté aux besoins nouveaux qui apparaissent régulièrement dans les projets de recherche. De plus, il est inévitable et même souhaitable pour un logiciel de cette envergure qu'il puisse être conçu en équipe. La modélisation adoptée devait donc également être modulaire, de manière à permettre aux membres de l'équipe de travailler séparément sur des parties spécifiques du programme et de combiner ensuite aisément ces modules.

Ces raisons font que notre choix s'est porté sur C++. Ce langage de programmation est Orienté-Objet **[1 ; 3]** et permet comme nous allons le voir de répondre aux attentes que nous venons d'exprimer. Il est de surcroît très puissant et dispose de nombreuses librairies mathématiques.

Eléments essentiels de la programmation OO

On parle souvent de la programmation par objets comme d'un paradigme, un ensemble de théories, standards et méthodes qui représentent une façon d'organiser les connaissances, c'est-à-dire une façon de voir le monde. Nous allons donc présenter les aspects essentiels de ce paradigme et nous verrons qu'il permet d'aborder un problème avec des méthodes très similaires à celles que nous employons dans la vie quotidienne.

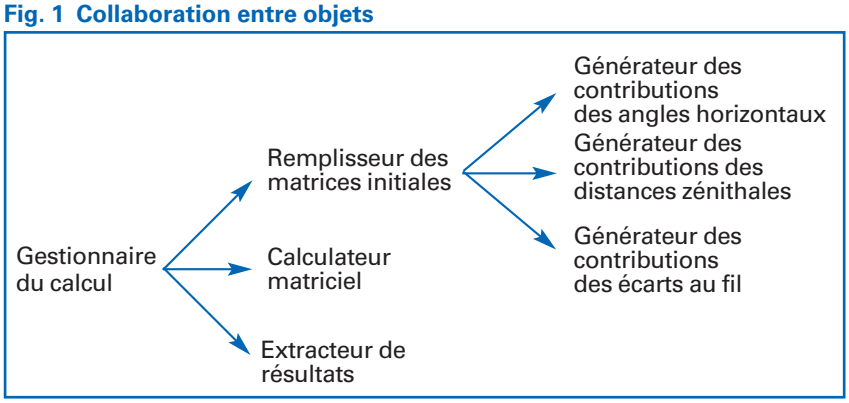
En programmation OO, l'action est déclenchée par la transmission d'un message à un agent (un objet) responsable de l'action. Le code du message indique l'action demandée ; on l'accompagne d'informations complémentaires (les arguments) nécessaires à son exécution.

Objets, messages et méthodes, attributs

En programmation OO, l'action est déclenchée par la transmission d'un message à un agent (un objet) responsable de l'action. Le code du message indique l'action demandée ; on l'accompagne d'informations complémentaires (les arguments) nécessaires à son exécution. Si l'agent accepte le message, il s'engage à répondre à la demande, et va pour ce faire appliquer une méthode. On peut faire l'analogie avec la démarche que nous employons souvent et qui nous amène à nous adresser à la personne dont les responsabilités la rendent susceptible de répondre à une certaine requête.

Notre logiciel de compensation fait par exemple intervenir un objet responsable de la gestion des étapes du calcul allant du dimensionnement des matrices initiales jusqu'à l'extraction des résultats finaux. Cet objet va lui-même à un certain moment faire appel à un autre objet responsable uniquement du remplissage des matrices initiales, lequel va à son tour collaborer avec des objets responsables chacun du calcul des contributions d'un type d'observation particu-

Fig. 1 Collaboration entre objets



lier à ces matrices. Un programme orienté-objet est donc structuré en termes de responsabilités.

A travers cet exemple apparaissent également les notions importantes de collaboration entre objets (fig. 1) et de masquage de l'information. En effet, selon le principe de délégation qui lui aussi nous est familier, un objet va très souvent s'adresser à d'autres objets qui vont effectuer des tâches, lui permettant ainsi de répondre au message qu'on lui a adressé. On parle de masquage de l'information parce que l'expéditeur du message n'a pas besoin de savoir par quels moyens sa demande sera réalisée.

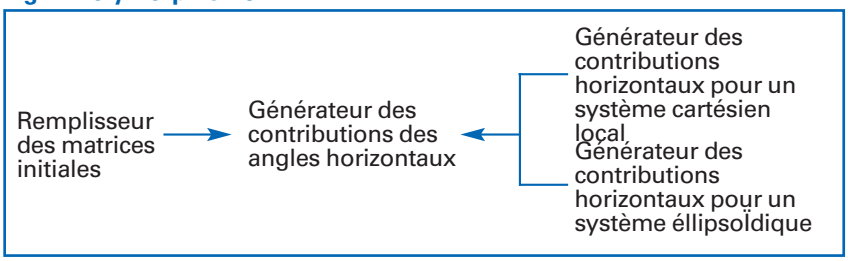
Les objets n'ont pas seulement des méthodes, mais ils ont aussi des attributs qui les caractérisent. Ainsi on retrouve dans notre programme des objets représentant des longueurs auxquels on peut envoyer des messages leur demandant de renvoyer leur valeur dans différentes unités. Ces objets doivent bien sûr renfermer un attribut numérique dans lequel est stocké la longueur représentée. Les objets sont donc définis par leur comportement (leur réponse aux messages qu'on leur adresse) ainsi que par leur état décrit dans leurs attributs.

2.2 Polymorphisme

On pourrait croire à ce stade qu'il y a peu de différences avec la programmation impérative en C ou Pascal dans lesquels on retrouve le masquage de l'information. En effet l'action y est déclenchée par un appel de procédure, qui pour répondre à la demande va elle-même faire appel à d'autres procédures. En fait l'envoi de message se distingue de l'appel de procédure parce qu'il a un receveur désigné. Plusieurs objets de nature différente sont susceptibles de répondre à une même demande (c'est la notion importante de polymorphisme) et l'interprétation, c'est-à-dire la sélection d'une méthode à exécuter en réponse à un message, peut varier selon le receveur. Comme le receveur désigné n'est généralement pas connu avant l'envoi du message, il n'est pas possible de déterminer à l'avance la méthode applicable. C'est pourquoi on dit qu'il y a une liaison différée entre le message et le code utilisé pour y répondre.

Dans notre logiciel on retrouve par exemple le polymorphisme au moment ou une observation est transmise à l'objet qui va placer dans les matrices des dérivées partielles et des poids les contributions de cette observation. Par exemple le calcul des contributions d'un angle horizontal sera différent si la compensation se fait dans un référentiel cartésien local ou si elle tient compte de l'ellipsoïde terrestre. Il y a donc un objet qui génère les contributions des angles horizontaux dans le cas du référentiel cartésien, et un autre auquel on fait appel dans le cas du référentiel ellipsoïdique. L'objet adéquat est mis en place au moment où dans les options de calcul le type de référentiel est sélectionné. Cependant cette distinction est inutile pour l'objet qui transmet les angles horizontaux à l'agent chargé du calcul des contributions. Cet objet sait uniquement qu'il collabore avec un agent responsable des angles horizontaux, apte à répondre aux demandes de calcul des contributions (fig. 2). Le code pour l'envoi du message est unique et donc simplifié, alors que les portions de code invoquées en réponse à ce message au moment de l'exécution du programme peuvent être distinctes.

Fig. 2 Polymorphisme



2.3 Classes et instances, héritage

Notre logiciel fait bien sûr intervenir des objets qui représentent les points en 3D nécessaires au calcul. Il serait très fastidieux d'écrire un code spécifique à chacun de ces objets, d'autant plus qu'on attend d'eux qu'ils aient le même comportement.

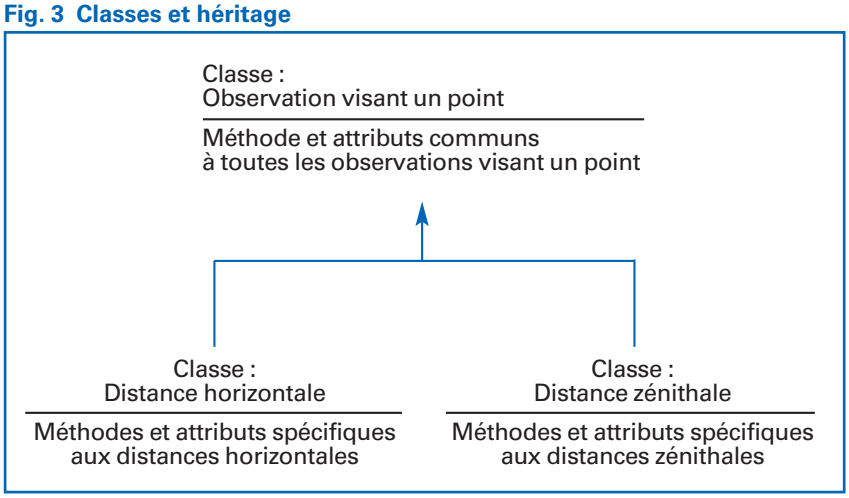
Cela nous amène à une autre notion essentielle de la programmation OO, celle de classes et instances. Tous les objets sont les instances d'une classe; la méthode invoquée par un objet en réponse à un message est déterminée par sa classe. Tous les objets d'une classe donnée utilisent la même méthode en réponse au même message et ont la même structure d'attributs.

De plus, des classes peuvent partager certaines de leurs caractéristiques. On peut considérer par exemple qu'une observation de distance horizontale et une observation de distance zénithale ont toutes les deux un point visé qu'on peut leur demander d'indiquer. Ces propriétés communes peuvent être décrites une seule fois dans le code grâce à la notion d'héritage. Les classes peuvent être organisées en une structure

Plusieurs objets de nature différente sont susceptibles de répondre à une même demande (c'est la notion importante de polymorphisme) et l'interprétation, c'est-à-dire la sélection d'une méthode à exécuter en réponse à un message, peut varier selon le receveur. Comme le receveur désigné n'est généralement pas connu avant l'envoi du message, il n'est pas possible de déterminer à l'avance la méthode applicable. C'est pourquoi on dit qu'il y a une liaison différée entre le message et le code utilisé pour y répondre.

hiérarchique. Une sous-classe hérite des propriétés d'une superclasse plus élevée dans l'arbre d'héritage. Ainsi, dans notre modèle, les classes représentant les distances horizontales et zénithales vont-elles toutes deux hériter d'une superclasse renfermant les propriétés communes à toutes les observations visant un point (fig. 3) , et le code décrivant ces propriétés peut n'être écrit qu'une seule fois.

Fig. 3 Classes et héritage



3. Synthèse

On dit souvent que la programmation par objets est une inversion de ce qui se faisait auparavant. Ce ne sont plus les procédures (les groupes d'instructions) qui ont la responsabilité d'arriver au résultat demandé en s'invoquant les unes les autres tout en transmettant les données nécessaires.

Dans l'orientation-objet ce sont les données elles-mêmes (les objets) qui sont actives et qui communiquent les unes avec les autres en s'adressant des messages.

On peut ainsi voir cette méthodologie comme une sorte de processus qui vise à créer une communauté d'aides qui assiste le programmeur dans la résolution d'un problème. Il suffit donc de décrire les différentes entités en jeu, de définir leur mode de communication, et de les exécuter. La programmation devient en fait une simulation.

Notre expérience avec ce logiciel de calcul nous permet d'attester que l'accent mis sur l'indépendance des composants individuels permet un développement incrémental et facilite le travail en équipe. Les unités individuelles sont définies, programmées et testées avant d'être regroupées en un système plus conséquent. Cet aspect modulaire facilite aussi spectaculairement la maintenance et la mise à jour du logiciel.

Très souvent les personnes qui ont goûté à la programmation OO n'aiment pas revenir à une programmation procédurielle. En effet, quand les programmeurs pensent en termes de comportement et de responsabilités des objets, ils apportent à leur travail toute la richesse de leur intuition, de leurs idées et de leur compréhension de la vie de tous les jours. C'est un processus mental beaucoup plus agréable. Nous espérons que l'aperçu que nous en avons donné dans cet article pourra inciter le lecteur à en faire l'expérience. ●

Références

[1] T. Budd. Introduciton à la programmation par objets. Addison-Wesley, 1992.

[2] J.P. Delahaye, I. Wilson, et al. CLIC, a multi-TeV e± collider. CERN-PS/99-062, 1999.

[3] www.cetus-links.org 18522 links on objects and components.

Quand les programmeurs pensent en termes de comportement et de responsabilités des objets, ils apportent à leur travail toute la richesse de leur intuition, de leurs idées et de leur compréhension de la vie de tous les jours. C'est un processus mental beaucoup plus agréable.