Chapitre 2

Le Contexte

L'historique

La grande majorité des codes de calcul du domaine scientifique a été développée en langage Fortran il y a entre quinze et trente ans.
Les codes aujourd'hui suivent des normes qualité de codage et de documentation. Dans ces années-là, ces normes étaient laissées à la discrétion des chefs de projet, développeurs et clients. Donc, suivant les cas, les codes étaient très bien écrits et documentés, d'autres beaucoup moins.
Les évolutions permanentes et le manque de documentation font que la connaissance du code peut s'être perdue au cours du temps, et les diverses interventions dans les développements successifs pèsent sur la logique des enchaînements et les compliquent souvent. Quant aux données et aux résultats de calcul, ils sont généralement mélangés et pas vraiment identifiés dans les codes. La personne qui gère ou utilise ces codes doit souvent repartir de zéro pour en comprendre le fonctionnement et se trouve confrontée à des difficultés pour les faire évoluer.
Ces inconvénients sont intrinsèques au langage Fortran lui-même qui ne contient pas de structure de données ni d'allocation dynamique de la mémoire. Des extensions du langage Fortran ont été écrites pour pallier ces inconvénients (directement ou avec un pré-compilateur), mais elles sont du domaine privé et doivent suivre les évolutions techniques.

La problématique

Comment conserver toute cette connaissance acquise et empêcher que ces codes ne deviennent obsolètes : les maintenir, les faire évoluer, les faire vivre et les intégrer dans les nouvelles technologies ?
L'enseignement en Faculté porte sur les langages Objet (Java, C++, ...) et le langage C ; le langage Fortran est peu répandu dans les programmes de l'enseignement purement informatique.
La solution d'une réécriture en langage objet est en général de trop grande envergure et nécessite de gros moyens financiers et humains. De plus les physiciens sont habitués au Fortran, d'apprentissage facile, car proche des mathématiques. Les interventions ponctuelles dans la programmation sont aisées et les temps d'exécution performants.

Notre solution

Dans ce contexte, la solution la plus efficiente consiste à garder le Fortran en l'encapsulant dans un langage Objet qui s'interface avec le C. Nous avons choisi le C++.
Le code Fortran est décomposé en modules Fortran indépendants intégrés dans une classe C++ que l'on baptisera « composant métier » C++/Fortran. L'utilisateur pourra continuer à travailler dans ses modules Fortran.
Ainsi on aboutira progressivement à une base de données de « composants métiers » utilisables par le code d'origine ou par d'autres applications.

La notion de « composant métier » C++/Fortran

Un composant est un programme réutilisable par plusieurs applications, indépendamment des langages et des plates-formes. Un composant est caractérisé par un nom, un service (ensemble d'interfaces composées d'opérations), des valeurs et des propriétés. Il a une identité et une interface bien définie. Il effectue des interactions avec d'autres composants via des connecteurs.
Le composant métier C++/Fortran n'est pas un véritable composant au sens objet, mais en possède les principales caractéristiques :

- une méthode de création (allocation dynamique),
- une méthode d'initialisation,
- une structure de données accessible de l'extérieur (paramètres d'entrée, sortie et constantes),
- une ou des méthodes d'utilisation,
- une documentation du composant,
- un ou des tests unitaires,
- une méthode d'écriture et de lecture du composant (persistance),
- une méthode de clonage du composant,
- une méthode d'impression du composant,
- un test global du composant,
- une méthode de destruction.

Les avantages

La restructuration du code Fortran en modules indépendants permet :

- une utilisation clarifiée : l'utilisateur à une vision exacte des fonctionnalités de chaque module ;
- une lisibilité accrue : identification des variables en entrées et sorties de chaque module ;
- une maintenance plus facile ;
- une possibilité de mettre en évidence des utilitaires communs à plusieurs codes.

Cette identification des variables d'entrées/sorties oblige à associer une documentation et des cas tests, d'où :

- une recapitalisation de la connaissance : l'intervention sur le code d'origine permet de reprendre en main la connaissance du code ;
- une documentation associée ;
- une base de cas tests unitaires.

L'intégration du module dans un langage objet permet :

- une meilleure gestion de la mémoire : la mémoire est gérée par le C++ ; il n'y a pas de duplication de la mémoire ;
- une meilleure visibilité des structures de données ;
- un test global du composant ;
- une réutilisation par héritage : on pourra hériter d'un composant ;

- une intégration du composant dans d'autres langages (Java, Python …) ;
- d'utiliser des outils de modélisation (UML) ;
- d'utiliser le composant dans un environnement distribué (CORBA, RMI …).

La méthodologie

Dans une première phase, nous allons décrire une méthodologie simplifiée. Le Fortran initial est très peu touché, il est composé d'un programme principal et de sous-programmes, et le composant métier C++/Fortran minimal. Cette méthode permet de continuer à développer entièrement en Fortran.
Cette méthodologie sera illustrée avec un exemple concret dans le chapitre « Mise en œuvre simplifiée ».
Dans la deuxième phase, la méthodologie est plus riche. Le Fortran est plus restructuré, découpé en fonctionnalités autonomes, et le composant métier C++/Fortran plus élaboré et réutilisable dans une autre architecture. On peut prendre l'exemple d'un programme constitué d'un « mailleur », d'un solveur et de post-traitements ; on va séparer ces trois fonctions afin de pouvoir les appeler globalement ou individuellement. On peut continuer à développer dans le code Fortran, mais seulement à l'intérieur du programme principal et de chaque sous-programme. Cette méthode permet un passage en douceur d'un développement Fortran à un développement C++. Cette méthodologie sera illustrée dans le chapitre « Mise en œuvre avancée ».
La méthodologie sera d'autant plus facile à appliquer si le code Fortran initial a suivi des règles de programmation et de documentation.