Méthodologie et environnement de développement orientés objets : de l'analyse mathématique à la programmation

S. Labbé, J. Laminie et V. Louvet

Laboratoire de Mathématique,
Analyse Numérique et E.D.P.,
Université de Paris Sud et C.N.R.S.,
Orsay, France

9 septembre 2003

Introduction

Le but de ce travail est la réalisation d'un cadre de programmation objet pour le calcul scientifique appliqué à la résolution d'équations ou de systèmes d'équations aux dérivées partielles. Les motivations qui nous ont conduits à concevoir ce projet sont diverses.En premier lieu, la programmation objet est moins directe que la programmation procédurale que permet les langages comme Fortran-77 et c. La complexité des infrastructures informatiques et celle des besoins des utilisateurs/concepteurs de logiciels font qu'il est impératif de changer la technologie de conception, pour prendre en compte de façon efficace des notions comme le travail et le développement coopératif, l'hétérogénéité des matériels... De plus, en tant que laboratoire universitaire, il s'ajoute les besoins d'outils pédagogiques pour la formation des étudiants, les besoins de pérénisation des codes de recherches, ...

La réalisation de codes de calcul scientifique dans la cadre d'un travail de recherche diffère de la situation classique qui demande juste la résolution du problème et l'obtention des quantités souhaitées par l'utilisateur final. La recherche dans le domaine du calcul scientifique se situe à de nombreux niveaux. Outre la résolution du problème, les numériciens travaillent également sur la mathématique des équations du problème, sur la méthodologie de résolution, sur les discrétisations mais aussi sur les outils de plus bas niveau comme l'algèbre linéaire par exemple. Nous nous permettons dans ce cahier des charges d'être relativement ambitieux, pour demander que la conception soit indépendante de la méthode de discrétisation et des techniques de résolutions.

Par ailleurs et d'un point de vue purement informatique, nous sommes conduit à travailler dans un environnement multi-platformes et multi-langages. L'analyse des résultats des codes, au niveau souhaité par les numériciens, implique la mise en oeuvre d'outils de développements, d'exploitation et de dépouillement.

La première section illustre la partie mathématique de ce projet, dont la mise en oeuvre permet de répondre au besoin concernant l'indépendance des objets mathématiques manipulés. La seconde rapporte des éléments de réflexion méthodologique comme aide à la conception objet. La troisième partie présente l'application de la méthodologie au problème mathématique objet décrit en première section.

1 Cadre théorique

L'objectif du projet est de résoudre de façon générique, c'est-à-dire quelle que soit la méthode de discrétisation, un système d'équation aux dérivées partielles. Dans ce paragraphe, nous décrivons

une représentation unifiée des méthodes de discrétisation usuelles (éléments finis, différences finies et volumes finis). Celle-ci repose sur des opérateurs de plongement et de restriction d'espaces fonctionnels vers des espaces de dimensions finies.

Supposons donc que l'on veuille traiter le problème suivant :

$$\text{soit} f \in \mathrm{W}_2, \text{ trouver } u \in \mathrm{W}_1 \text{ tel que } \mathcal{P}u = f, \tag{1}$$

oùW_1 et W_2 sont deux espaces fonctionnels (de manière trés générale, on peut par exemple dire que ce sont des espaces de Banach séparables), $\mathcal{P}$ représente un opérateur linéaire ou non, u la solution, et f les forces extérieures. Afin de résoudre ce système, il est nécessaire de définir un problème approché c'est-à-dire de réécrire le système dans des espaces de dimension finie sous la forme

$$\text{soit} f_h \in \mathrm{W}_{2,h}, \text{ trouver } u_h \in \mathrm{W}_{1,h} \text{ tel que } \mathcal{P}_h u_h = f_h, \tag{2}$$

où $\mathrm{W}_{1,h}$ et $\mathrm{W}_{2,h}$ sont deux espaces de dimension finie, u_h la solution approchée et f_h l'approximation des forces extérieures.

Considérons les opérateurs de plongement $\mathrm{P}_h^* : \mathrm{W}_{1,h} \mapsto \mathrm{W}_1$ (resp. $\mathrm{Q}_h^* : \mathrm{W}_{2,h} \mapsto \mathrm{W}_2$) et de restriction $\mathrm{P}_h : \mathrm{W}_1 \mapsto \mathrm{W}_{1,h}$ (resp. $\mathrm{Q}_h : \mathrm{W}_2 \mapsto \mathrm{W}_{2,h}$). On obtient alors le diagramme suivant :

$$\begin{array}{ccc}
\mathrm{W}_{1,h} & \xrightarrow{\mathcal{P}_h} & \mathrm{W}_{2,h} \\
\mathrm{P}_h \uparrow & & \downarrow \mathrm{Q}_h^* \\
\mathrm{W}_1 & \xrightarrow{\mathcal{P}} & \mathrm{W}_2 \\
\mathrm{P}_h^* \uparrow & & \downarrow \mathrm{Q}_h \\
\mathrm{W}_{1,h} & \xrightarrow{\mathcal{P}_h} & \mathrm{W}_{2,h}
\end{array}$$

On peut ainsi définir l'opérateur discret $\mathcal{P}_h$ en fonction des opérateurs de restriction et de plongement et du problème continu $\mathcal{P}$:

$$\forall h \in \mathbb{R}^p, \mathcal{P}_h = \mathrm{Q}_h \circ \mathcal{P} \circ \mathrm{P}_h^*,$$

Ces opérateurs de discrétisation sont construits de façon à vérifier un certain nombre d'hypothèses permettant d'assurer qu'ils sont candidats pour être de "bonnes" discrétisations.

Hypothèse 1.1 *On suppose que les opérateurs P_h, Q_h, P_h^*, Q_h^* sont tels que :*
(i) Pour tout h dans $\mathbb{R}^p$, P_h et Q_h sont des opérateurs linéaires continus.
(ii) On a :

$$\begin{aligned}
\forall u \in W_1, \lim_{|h|\to 0} ||P_h^* \circ P_h \; u - u||_{W_1} &= 0, \\
\forall u \in W_2, \lim_{|h|\to 0} ||Q_h^* \circ Q_h \; u - u||_{W_2} &= 0, \\
\forall u \in W_{1,h}, \lim_{|h|\to 0} ||P_h \circ P_h^* \; u - u||_{W_{1,h}} &= 0, \\
\forall u \in W_{2,h}, \lim_{|h|\to 0} ||Q_h \circ Q_h^* \; u - u||_{W_{2,h}} &= 0.
\end{aligned}$$

Dans le cas, fréquent, où $\mathrm{W}_{1,h}$ est inclus dans W_1 et $\mathrm{W}_{2,h}$ est inclus dans W_2, les deux dernière lignes du point *(ii)* de l'hypothèse (1.1) sont automatiquement vérifiées.

On démontre alors que les discrétisations ainsi construites sont automatiquement consistantes, la stabilité quand à elle dépend fortement, bien entendu, du problème continu.

Ainsi, afin d'illustrer ceci, nous montrons ici deux exemples d'opérateurs : les éléments finis P1 et les différences finies.

En éléments finis, l'espace $\mathrm{W}_{1,h}$ sera celui engendré par les fonctions de base $(\varphi_i)_{i=1,..,N}$, où N est le nombre de sommets de la triangulation (on notera pour la suite que les sommets numérotés entre 0 et N_{int} sont à l'intérieur du maillage et les autres sur le bord du maillage). Dans notre exemple, les fonctions de base sont de type P1, tandis que l'espace $\mathrm{W}_{2,h}$ est celui des fonctions constantes par morceaux sur les cellules du maillage. Ainsi, les opérateurs P_h et P_h^* sont définis comme suit :

$$\forall u \in \mathrm{W}_1,\ \mathrm{P}_h(u) = (\Phi, u).\Phi,$$
$$\forall u_h \in \mathrm{W}_{1,h},\ \mathrm{P}_h^*(u_h) = M^{-1}(\Phi, u_h).\Phi,$$

où $(.,.)$ désigne le produit scalaire dans L^2, $\Phi = (\varphi_1, ..., \varphi_N)^t$ et $M = (\Phi, \Phi^t)$.

En ce qui concerne la méthode des différences finies, on adopte ici une technique de colocation qui est présentée, par soucis de "légèreté" de l'écriture, sur le maillage régulier du carrée $[0,1]\times[0,1]$ dont les noeuds sont les points $x_i = (p\ h, l\ h)^t = (x_{i,1}, x_{i,2})^t$ pour $i = (p,l)^t$ et p et l variant entre 0 et N $(h = 1/N)$; de plus, on pose $\omega_i = [(p-1/2)h, (p+1/2)h]\times[(l-1/2)h, (l+1/2)h]\cap[0,1]\times[0,1]$. Considérons la notation suivante

$$\forall i \in \{0, ..., N\} \times \{0, ..., N\},$$
$$P_i(x) = ((x_1 - x_{i,1})^2, (x_2 - x_{i,2})^2, (x_1 - x_{i,1}), (x_2 - x_{i,2}), 1)^t.$$

Alors on montre qu'il existe une matrice inversible A carré d'ordre 5 (qui ne dépend pas du point dans le cas exposé ici des maillages réguliers), telle que, pour tout élément u de W_1, $\mathrm{P}_h u$ coïncide avec u aux points du maillage. L'opérateur $\mathrm{P}_h u$ s'exprime sous la forme :

$$\forall u \in \mathrm{W}_1, \forall i \in \{0, ..., N\} \times \{0, ..., N\}, \forall x \in \omega_i,\ (\mathrm{P}_h(u))(x) = A^{-1} Y_i . P_i(x),$$

où Y_i est le vecteur des valeurs de u sur la maille i et les quatres mailles voisines. Le relèvement $\mathrm{P}_h^* u_h$ d'un élément u_h de $\mathrm{W}_{1,h}$ est quand à lui la fonction de W_1 la plus proche u_h au sens de la norme de W_1 et coincidant avec u_h aux points du maillage.

Ce travail est présenté dans [6], dans lequel se trouve, sur le même principe, la réécriture des méthodes de volumes finis dans le même cadre.

Les opérateurs pour les cas des méthodes des éléments finis P1 et des différences finies sur un maillage régulier étant définis, nous allons voir comment exploiter ces écritures dans le cadre d'une programmation orientée objets de ces méthodes. Ces deux méthodes serviront d'exemple d'illustration dans la suite.

2 Méthodologie pour la conception orientée objet du projet

La conception générale du projet s'appuie sur une approche objet de la problématique globale comprenant la partie calcul scientifique mais aussi l'ensemble des outils d'instrumentation et de développement associés. Elle nécessite la mise en oeuvre d'une méthodologie adaptée à cet objectif.

Toute méthodologie peut être considérée comme la complétion d'un formalisme et d'un processus. Dans le cadre de ce projet, notre méthodologie, basée sur le formalisme UML (Unified Modeling Language) [2], est inspirée du processus UP (Unified Process) [1]. UML est un support graphique et formel pour la modélisation d'un logiciel (notations standardisées avec une sémantique précise). UP définit un ensemble d'étapes permettant de mener à bien l'élaboration d'un logiciel :

1. conceptualisation des besoins,
2. analyse et réalisation des fonctionnalités,
3. prise en compte de l'architecture et des contraintes techniques,
4. conception.

On présente ici une vue simplifiée de la démarche de conception.

2.1 Conceptualisation

La première étape du processus consiste à reformuler les besoins et à indiquer de façon claire les fonctionnalités attendues. Pour expliciter ces besoins, exprimés dans la première partie de ce document, on peut utiliser le formalisme des cas d'utilisation d'UML (Use Case) [5]. Chaque fonctionnalité du logiciel correspond donc à un cas d'utilisation. Le scénario principal de chaque cas d'utilisation décrit le déroulement d'un ensemble d'actions permettant d'atteindre l'objectif, c'est à dire de réaliser la fonctionnalité associée.

Conformément aux besoins exprimés, plusieurs fonctionnalités et donc plusieurs cas d'utilisation peuvent être définis :

1. Résoudre un problème de calcul scientifique,
2. Suivre la résolution d'un problème de calcul scientifique,
3. Contrôler la résolution d'un problème de calcul scientifique,
4. Consulter la documentation d'un code de calcul scientifique,
5. Coupler des codes de calcul scientifique.

Le premier cas d'utilisation concerne la partie mathématique du projet. Les autres correspondent plutôt à l'environnement d'exploitation des calculs.

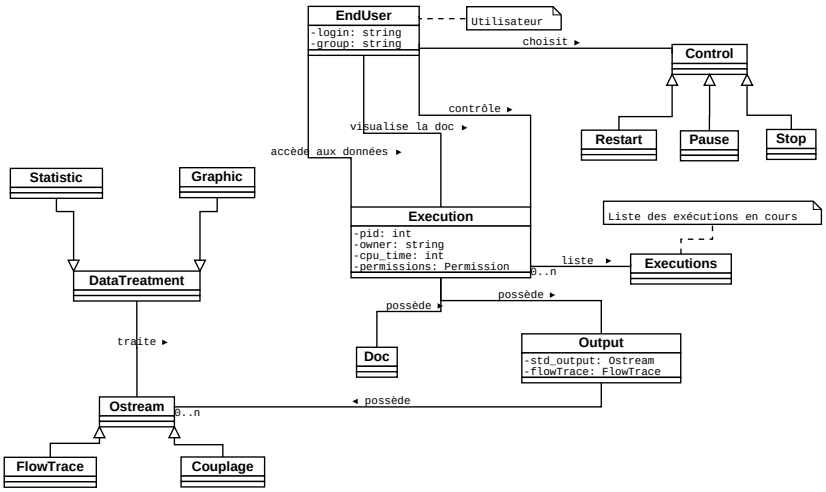
L'élaboration du scénario principal du premier cas d'utilisation fait appel aux connaissances métier relatives à notre domaine d'étude c'est-à-dire de façon générale le calcul scientifique. Cette partie est détaillée en section 3.

La description des scénarii principaux des autres cas d'utilisations permettent de produire un certain nombres de substantifs sur lesquels pourront se baser les classes d'analyse.

2.2 Analyse

La phase d'analyse commence par recenser et détailler l'ensemble des substantifs apparaissant dans la phase de conception. Il s'agit de représenter tout ce qui est manipulé par le logiciel. L'établissement d'un glossaire est complété par un diagramme de classe UML.

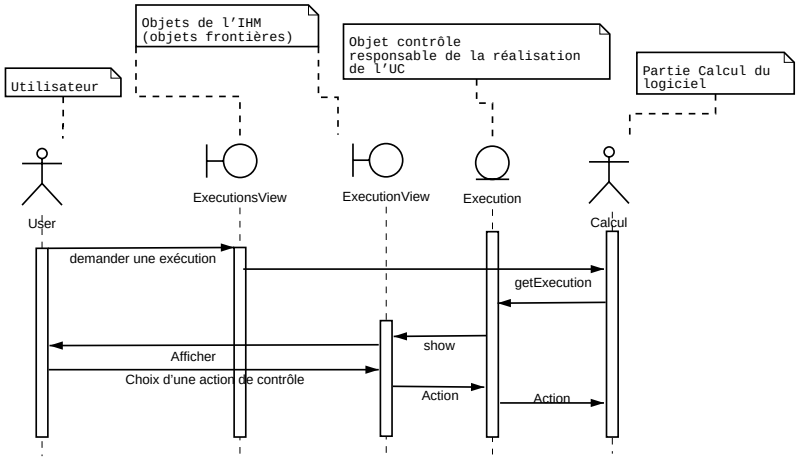
Le diagramme suivant illustre de façon très simplifiée les entités qui sont issues de l'étude des cas d'utilisation de l'instrumentation.



Cette première partie est généralement appelée analyse du domaine. Les classes ainsi définies servent pour l'analyse applicative qui consiste à réaliser les cas d'utilisation grâce à des diagrammes de séquences.

La réalisation, sous forme simplifiée, du cas d'utilisation 3, est illustrée ci-dessous.

On considère comme pré-condition de ce cas d'utilisation que l'utilisateur est identifié au sein du logiciel, qui lui permet l'accès aux données. Cet UC inclut des objets de l'IHM.



Il s'agit de définir les responsabilités de chaque classe et d'identifier les objets de contrôle qui porte en général la responsabilité de réalisation des cas d'utilisation.

2.3 Architecture

Cette étape permet de prendre en compte l'architecture et les contraintes techniques. On peut considérer que le projet comporte trois grandes parties :

- la présentation, c'est-à-dire l'IHM,
- les traitements, c'est-à-dire le calcul lui-même,
- les données, c'est-à-dire tout ce qui doit être conservé pour le suivi du calcul, l'exploitation des résultats, ...

Chacune d'elles nécessitent des besoins techniques particuliers qu'il est important de prendre en compte :

- L'IHM doit être multiplateforme,
- le calcul demande des performances machines importantes,
- les données doivent être toujours à disposition.

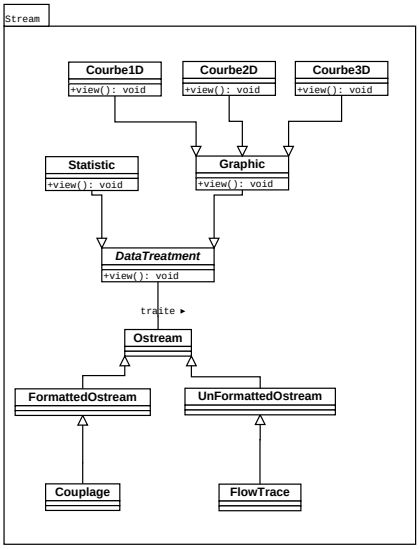
Pour bénéficier d'une modularité optimale, il faut définir des contrats d'architecture (représenter par des interfaces) sur chacune des parties de l'architecture. Ainsi, chaque section du projet devient indépendante des autres puisqu'elles utilisent leur contrats respectifs. Leur implémentation peut donc changer sans conséquence pour le reste du logiciel.

Les choix techniques effectués pour la réalisation du logiciel et basés sur cette analyse sont expliciter en 2.5.

2.4 Conception

La conception proprement dite consiste à amalgamer les objets et classes d'analyse dans l'architecture technique.

Cela conduit à réaliser des encapsulations (package de classes ayant entres elles des relations fortes - héritage, aggrégation -) intégrées aux encapsulations architecturales définies en 2.3. Le diagramme suivant illustre de façon simplifiée (Seules les classes relatives au flux sortants sont indiquées) le découpage de la partie flux de données de l'instrumentation, qui concerne de manière globale les entrées/sorties générées et utilisées par le logiciel, ainsi que les traitements qui y sont associés.



Une fois les packages et les classes bien définis, il faut y associer des besoins techniques précis (cycle de vie des objets, construction, destruction, partage ...) qui conduiront à l'implémentation proprement dite. L'utilisation des patrons architecturaux, qui sont des modèles de conception réutilisable, est particulièrement importante pour cette phase du développement [3].

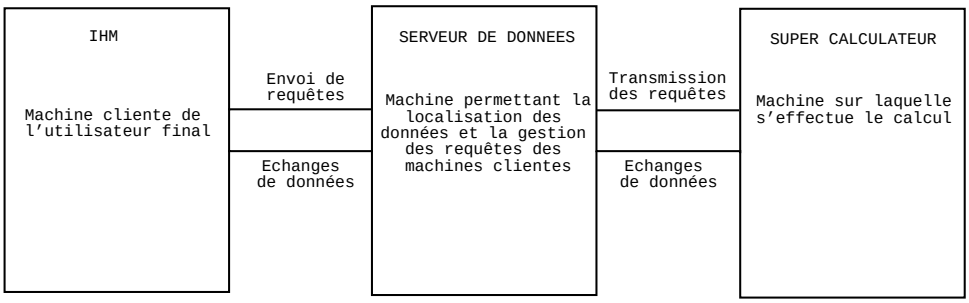
Cette étape nous permet de développer une première version du logiciel.

Le cycle complet de conception est itératif et s'améliore et s'enrichit à chaque itération. Ainsi, l'ajout d'une nouvelle fonctionnalité ne remettra pas en cause ce qui a déjà été développé, mais s'intégrera naturellement à l'ensemble du projet.

2.5 Choix techniques et implémentation

Les choix techniques réalisés sont basés sur l'analyse de l'architecture du projet présenté en 2.3. Une structure architecturale de type 3 tiers paraît adaptée à une utilisation rationnelle des moyens informatiques. Ainsi, il est cohérent de prévoir, pour l'implémentation du logiciels les points suivants :

- L'IHM s'exécute sur la machine de l'utilisateur final. Pour assurer un portage maximal, le choix du langage Java paraît adapté, de même qu'une utilisation du langage HTML pour la visualisation de la documentation des codes. Afin d'assurer une mise à jour permanente, celle-ci est intégrée aux sources et générée à la volée grâce à un script perl.
- Le calcul est effectué sur une machine dédiée de type supercalculateur. Le langage d'implémentation doit être performant et objet, d'où le choix actuel du C++.
- Les données sont distribuées aux machines d'exploitation. Celles-ci doivent donc avoir connaissance de la localisation du stockage de ces données. La mise en place d'un modèle d'architecture distribuée a donc été nécessaire. Le middelware CORBA [4] nous a semblé le plus adapté compte tenu de notre environnement de travail. Un serveur CORBA fait donc le lien entre le lieu de stockage des données et la machine sur laquelle elles sont exploitées.



3 Structure des classes et présentation d'un exemple

Comme l'explique la méthodologie de conception de la section précédente, la clef de l'analyse objet est de bien discerner les structures de données indépendantes pouvant avoir leur propre existence et comportement. Ainsi, dans un problème classique de la forme donnée par (1) avec une discrétisation (2), la première remarque est que le système continu existe indépendemment du système discret, la réciproque étant fausse. De même, on peut appliquer un raisonnement identique entre le système discret et les systèmes algébriques qu'il génère.

Nous venons donc de définir trois couches de classes : le continu, le discret et l'algébrique. Ces couches peuvent être associées à trois métiers différents : l'analyse numérique, la construction de schémas de discrétisation et la résolution de grand systèmes linéaires. Enfin, entre ces couches, afin de conserver la souplesse de la structure, il faut imposer des transferts d'adresses d'informations et non de données à proprement dites. D'autre part, une couche connaît toutes les couches inférieures mais ignore les couches supérieures. Cette structure peut être illustrée grâce au diagramme (1).

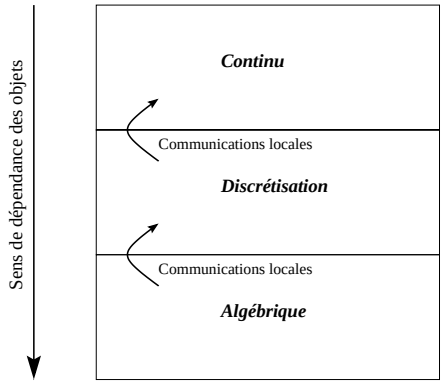


FIG. 1 – Structure en couches ou en métiers du code.

Chacune de ces couches comporte une structure de classes. Nous la présentons en parallèle de l'exemple que nous nous sommes fixés : l'équation de Laplace :

$$\begin{cases} \triangle u = f, \text{ dans } \Omega, \\ u = g, \text{ sur } \partial\Omega. \end{cases} \tag{3}$$

3.1 Structure générale des couches

Chaque couche correspond donc à la résolution d'un problème donné. Au substantif problème, sont associés les substantifs opérateurs, variables et domaines. Par conséquent, à chaque niveau de programmation, nous définissons un objet de type classe *problème*. Cette classe est l'élément distributeur d'informations dans la couche. Il est, en général, unique par couche. Il est lié par aggrégation propriétaire à la classe *problème* de la couche précédente et possède l'objet instancié à partir de la classe *problème* de la couche suivante. La durée de vie d'un *problème* est donc limitée à la durée de vie du *problème* de la couche précédente s'il existe.

Les deux autres objets qui apparaissent lors de l'analyse sont les objets *variables* et *opérateurs*. Les premiers contiennent par exemple les solutions des problèmes, des conditions initales ..., les suivants sont, comme leur nom l'indique, des opérateurs agissant sur une ou plusieurs variables et restituant une ou plusieurs variables. La plupart du temps, les *variables* et *opérateurs* sont liés au problème par une aggrégation, propriétaire ou non, mais sont également composants de l'objet de philosophie identique de la couche supérieure et possédent un objet de philosophie identique de la couche inférieure.

Enfin, on trouve les objets *domaines*, qui regroupent les informations éventuelles sur la topologie continue ou discrète des domaines sur lesquels les problèmes sont résolus.

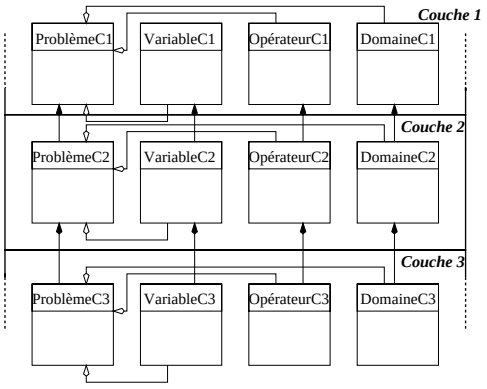


FIG. 2 – structure de classes génériques internes aux couches.

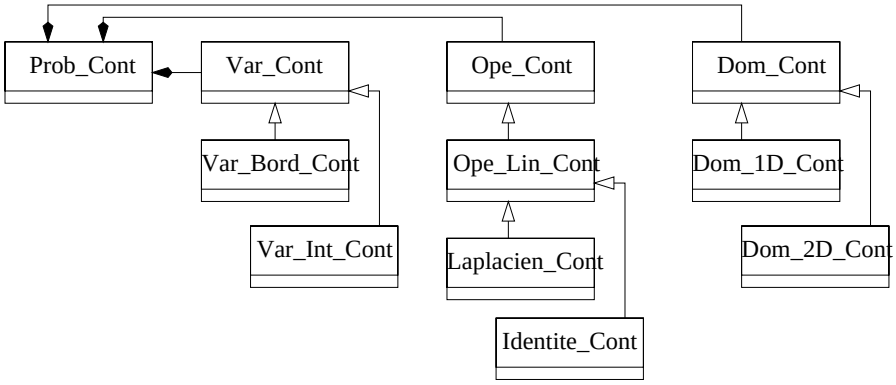


FIG. 3 – Structure des classes dans la couche "Continu".

Le schéma (2) illustre cette approche en utilisant les conventions UML. Les flèches à pointe diamant noires indiquent les agrégations propriétaires (la classe pointée possède un ou plusieurs objets de la classe pointant, le crée et le détruit). Les autres flèches, à pointe diamant blanches, représentent les agrégations non propriétaires (la classe pointée ne possède qu'une référence d'un ou plusieurs objets de la classe pointant).

3.2 Couche "Continu"

Considérons maintenant plus particulièrement la couche "continu". Le problème doit être relié à trois classes. Nous sommes ici dans la première couche de notre empilement. Les relations entres classes de la figure (2) à l'intérieur de la couche sont donc de type propriétaire. Le *Problème* déclaré au niveau continu possède donc tous les objets de toutes les couches. Son instanciation entraîne donc la création de tous les autres objets et sa destruction correspond à la fin du programme.

Pour traiter notre exemple de l'équation (3), nous allons spécialiser les classes de base citées plus haut, excepté, en général, les classe de type *Problème*.

Les *variables* sont de deux types, les unes définies sur le bord du domaine et les autres à l'intérieur du domaine. Cela amène naturellement à spécialiser également le domaine qui est une agrégation de domaine dont l'un est de type "bord" et l'autre de type "volume". Enfin, les opérateurs que nous appliquons sont linéaires, principalement le laplacien et l'identité. Ils agissent sur deux variables, l'une définie sur le bord du domaine et l'autre définie sur l'intérieur du domaine et renvoient deux variables de même type. Le schéma (3) représente le diagramme de cette hiérarchie.

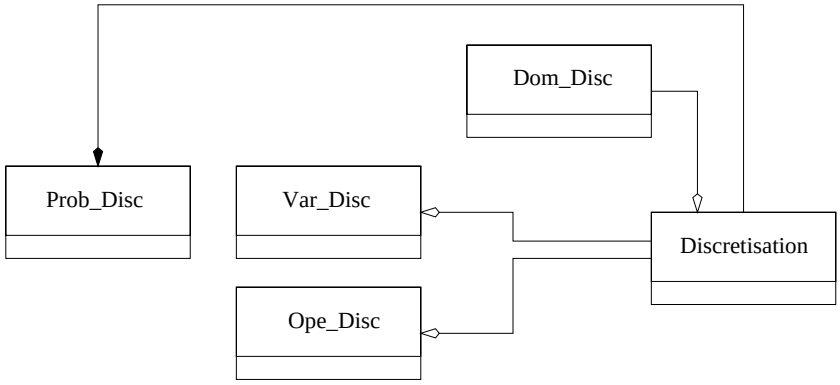


FIG. 4 – Lien entre la discrétisation et les autres classes de la couche "Discrétisation".

3.3 Couche "Discrétisation"

Dans la couche "Discrétisation", on retrouve l'équivalent de toutes les classes présentées pour la couche "Continu". Par contre, on voit apparaître un package supplémentaire contenant des classes décrivant le fonctionnement des releveurs et opérateurs permettant de passer des espaces de dimension finie aux espaces de dimension infinie et inversement. Ce package est accessible via une interface *Discrétisation* qui se spécialise éventuellement en d'autre classes (pour notre cas, celle des éléments finis P1 et des différences finies sur un maillage régulier). Elle est agrégée propriétairement à la classe *Problème* de la couche discrète et possède la référence sur le domaine discret (figure (4)). Les opérateurs, variables et problèmes de la couche "Discrétisation" n'obtiennent d'informations sur le maillage qu'au travers du filtre de la méthode. En effet, un même maillage peut être utilisé pour les éléments finis P1 ou P2 mais les degrés de liberté, par exemple, ne sont pas les mêmes.

Considérons la classe *Dicrétisation* : quelles sont les fonctionnalités attendues des instances (polymorphes) de cette classe ? Ils doivent tout d'abord pouvoir décrire le maillage, c'est-à-dire :

- ✓ parcourir les mailles,
- ✓ parcourir les degrés de liberté,
- ✓ repérer les bords du domaine,

mais ils doivent également permettre de :

- ✓ calculer "localement" (sur une maille) l'identité,
- ✓ calculer "localement" (sur une maille) le laplacien,

pour des fonctions définies dans l'espace de dimension finie sur lequel est projeté le problème continu. Jusqu'ici, nous n'avons toujours pas évoqué la méthode de discrétisation choisie. En effet, avec cette méthodologie, il est aisé de changer la méthode sans pour autant avoir à modifier la nature des *opérateurs*, des *variables* ou encore de *problème*, que ce soit au niveau "Continu" ou au niveau "Discrétisation". Ainsi, la classe *Discrétisation* est spécialisée, dans notre cas, en deux classes *EFP1* et *DFMR*, correspondant respectivement aux éléments finis P1 et aux différences finies sur un maillage régulier. Le polymorphisme permet alors de manipuler l'objet réel par l'intermédiaire d'une référence générique de type *Discrétisation*.

Prenons l'exemple du calcul du laplacien. Quand l'opérateur discret veut créer son équivalent algébrique, il va créer une matrice. Cette matrice va être assemblée en construisant des sous-matrices , élément du maillage par élément du maillage. Ainsi, pour les éléments finis, la fonction retournera la matrice :

$$A_k = -(\nabla\varphi_i, \nabla\varphi_j)_{(i,j)\in\mathcal{V}_k},$$

pour tous les élément k du maillage et où l'on note $\mathcal{V}_k$ l'ensemble des indices des degrés de liberté dépendant de k.

En ce qui concerne les différence finies, la matrice est simple car indépendante de la maille. Sur les point intérieurs, elle s'écrit :

$$A = \frac{1}{h^2}(1, 1, -4, 1, 1),$$

les coordonnées des sommets correspondant du maillage sont, comme pour les éléments finis disponibles par le biais de la classe *Discrétisation*.

3.4 Couche "algébrique"

Dans la couche algébrique, la plus simple, la structure est identique à celles des couches supérieures, excepté la disparition de la notion de domaine géométrique. Par contre, un package, trés similaire dans l'esprit à celui de la discrétisation, permet de gérer de la façon la plus transparente possible les problème de stockage.

4 Conclusion

Cette étude a été menée dans le laboratoire de mathématique de l'Université Paris Sud, au sein de l'équipe Analyse Numérique et EDP, dans le cadre d'un groupe de travail (http ://www.math.u-psud.fr/˜ gtoocs). Ce projet est actuellement appliqué sur le terrain par des chercheurs (permanents ou doctorants) pour développer leurs propres codes de calculs en utilisant une philosophie de méthodologie et de développement orientée objet. Une analyse détaillée pourra être trouvée dans [7].

Références

[1] P. Roques et F. Vallée. *UML en action*. Eyrolles, 2000.

[2] P.A. Muller et N. Gaertner. *Modélisation objet avec UML*. Eyrolles, 2000.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, 1999.

[4] M. Henning and S. Vinoski. *Advanced CORBA Programming with C++*. Addison Wesley, 1999.

[5] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object Oriented Software Engineering : A Use Case Driven Approach*. Addison Wesley, 1992.

[6] S. Labbé. Discrétisation des équations aux dérivées partielles : Analyse numérique orientée objets. En préparation.

[7] S. Labbé, J. Laminie, and V. Louvet. Méthodologie et environnement de développement orientés objets : de l'analyse mathématique à la programmation. Technical Report RT 2001-01, Université Paris-Sud, Laboratoire de Mathématique, 2002.