Une approche orientée objet avancée en Java pour le calcul en mécanique

Dominique Eyheramendy

Une approche orientée objet avancée en Java pour le calcul en mécanique

Dominique Eyheramendy

Equipe Modélisation Mathématique et Calcul Scientifique Hautes Performances pour l'Environnement, Institut Camille Jordan, UMR 5208, CDCSP – ISTIL, Université Lyon 1
15 Boulevard Latarjet, 69622 Villeurbanne Cedex
eyheramendy@cdcsp.univ-lyon1.fr

RÉSUMÉ. *Le développement de code éléments finis pour les simulations complexes s'avère être une activité souvent difficile. Cela représente cependant une étape importante dans le développement d'un outil de simulation. Depuis une quinzaine d'année, les approches orientées objet ont apporté à la mécanique des solutions informatiques pour le développement de codes éléments finis. Dans le contexte général de la résolution de problèmes complexes multi-physiques, à échelles multiples, sur des systèmes informatiques très hétérogènes, il nous apparaît important de poser une nouvelle fois le problème de la complexité. L'approche que nous proposons, basée sur un concept Java, tend à permettre une gestion globale et unifié de la complexité des codes éléments finis en mécanique : physique du problème, modèle éléments finis, algorithmes (parallèles par exemple), matériel informatique, réseau… Dans ce papier, on propose une première approche pour une forme avancée de code éléments finis en Java et on en montre la validité.*

ABSTRACT. *The development of finite elements codes for complex simulations is a tough time-consuming task. This still represents an important step in the elaboration of a computational tool. The object-oriented paradigm has been successfully applied to the finite elements method in mechanics for 15 years. Considering the multiple complexity for modern simulation tools -multi-physics, multi-scale, multi-processors systems-, it seems important to us to address again the problem of code complexity. The approach we propose is based on the Java environment. It leads to a global and unified management of the complexity of finite elements codes in mechanics: physical problem, finite elements model, numerical algorithms, (e.g. parallel), hardware, network… In this paper, we propose an advanced O.O features in Java for F.E. and we give the proof of the feasibility.*

MOTS-CLÉS : *Java, Programmation orientée objet, éléments finis, Java, plasticité J_2, parallélisme.*

KEYWORDS: *Java, Object-oriented programming, finite elements, J_2 plasticity, parallel computing.*

1. Introduction

Une approche avancée de structuration de codes éléments finis en vue du déploiement sur systèmes hétérogènes est présentée. Le problème crucial du choix du support logiciel pour le développement d'applications complexes de simulation en mécanique est posé. D'un point de vue technique, c'est le langage C++ qui est le plus utilisé à quelques rares exceptions prés. Il faut noter que pour les approches objet un grand nombre de langages ont été testés depuis le début de l'objet en mécanique : LISP, Smalltalk, CTalk, Eiffel, ADA, C++. Aujourd'hui, Java représente un certain intérêt pour l'industrie (Ginsberg et al., 2000). Des projets d'envergure se basent sur cette approche pour effectuer des calculs, par exemple le code CartaBlanca (calcul multiphasique avec chaleur) développé au Laboratoire National de Los Alamos-USA (Padial-Collins et al., 2004) ou le projet OASIS (application d'électromagnétisme) de l'INRIA (Baduel et al., 2004). Ces approches sont plutôt orientée informatique. On peut également noter les plates-formes de développement simulation/coupleurs de code du type de PALM au CERFACS ou SALOME à l'EDF basés sur des technologies classiques multiples : Fortran/MPI pour le premier et C++/Corba/Python principalement pour le second. Nous proposons dans ce travail une approche basée sur une technologie, Java, capable d'intégrer dans un concept unique (contrairement aux exemples précédents) tous les ingrédients nécessaires à la simulation numérique sur architectures hétérogènes. Dans cette communication, nous nous intéresserons à deux aspects de Java permettant de gérer la complexité des codes éléments finis en mécanique : gestion de la robustesse de code par le concept d'interface appliqué au matériau, et gestion de processus multiples pour le parallélisme pour une application de décomposition de domaine.

2. Un exemple de structuration objet avancé en Java appliqué au modèle de matériau

2.1. Notion d'interface

Une interface peut être définie comme une classe abstraite pure (Stoustrup, 2000), c'est à dire une classe qui ne peut pas être instanciée, mais qui permet de définir un type. Une interface permet donc de ne spécifier que des fonctionnalités, et ne contient que méthodes abstraites (en pratique des prototypes de méthodes). Les interfaces peuvent être hiérarchisées. Et c'est donc la classe qui permet d'implémenter l'interface, c'est à dire d'implémenter les méthodes spécifiées dans l'interface. Une classe peut ainsi implémenter plusieurs interfaces. Une instance d'une telle classe peut donc être typée soit comme la classe elle-même, soit comme l'une de ces interfaces. On trouvera plus de détails dans (Flanagan, 2002).

2.2. Un modèle objet « matériau » en Java pour la plasticité J_2

Dans l'élaboration d'un modèle numérique pour une loi de comportement en mécanique, on peut dissocier la loi de comportement de l'algorithme numérique qui permet de l'intégrer. On considère ici un matériau élastique parfaitement plastique J_2. L'intégration de loi de comportement est réalisée au niveau local, c'est à dire à chaque point de Gauss suivant un algorithme de type retour radial (Simo et al., 1998). La difficulté consiste à identifier les données propres à l'algorithme, les données liées au comportement particulier, et les données du matériau. La mise en forme objet de ce problème repose sur 3 objets : un objet matériau (gestion des caractéristiques physiques du matériau pour un comportement donné), un objet comportement (calcul de l'opérateur élastoplastique tangent, calcul de la correction plastique…), un objet intégrateur (algorithme d'intégration numérique). Pour chacun de ces objets, une classe générique abstraite définit le comportement général, et des sous-classes définissent les comportements particuliers. La hiérarchie de base pour ce problème est donnée Figure 1 : classes de base Material, Behavior et Integrator et sous-classes respectives pour le modèle étudié MisesPerfectlyPlastic, LinearElasticPerfectlyPlastic et RadialReturnAlgorithm. Ce modèle peut être étendu à des comportements beaucoup plus complexes (multi-surfaces par exemple pour les bétons ou les matériaux granulaires en général). Il pourrait être implanté dans d'autres langages à objets comme le C++ par exemple. Il d'ailleurs relativement proche dans l'esprit de celui de (Foerch, 1996).

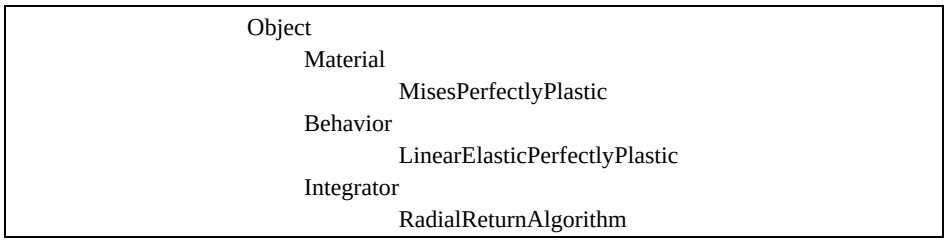


Figure 1. *Modèle objet de matériau en numérique*

2.3. Spécification d'intégrabilité de modèle de comportement

Un algorithme d'intégration ne peut être utilisé pour un comportement donné que si celui est validé des considérations théoriques et par des tests. La question qui se pose alors est de ne pouvoir utiliser un type d'intégrateur numérique que pour des lois de comportement donnés. L'idée est de définir une hiérarchie de spécifications dont les définitions sont liées aux intégrateurs. Dans le cas présent, on souhaite que l'algorithme de retour-radial permette l'intégration du modèle de plasticité parfaite développé, et seulement de celui-ci. L'implémentation objet repose sur la création d'une hiérarchie d'interfaces permettant de spécifier si un comportement peut

utiliser l'intégrateur considéré. Ici, on définit deux interfaces: l'interface générale de intégrateur (Integrable, littéralement : « qui peut être intégré ») qui impose aux classes qui l'implémentent d'avoir deux méthodes liées à tous les algorithmes d'intégration, et l'interface spécifique pour l'algorithme du retour-radial (RadialReturnIntegrable littéralement : « qui peut être intégré par le retour-radial ») qui impose le calcul du correcteur plastique. Ainsi, on obtient une implémentation naturelle et généralisable du concept de matériau grâce au concept objet, et une implémentation sûre grâce aux concepts objet étendus en Java. On trouvera les détails de l'implémentation dans (Eyheramendy, 2004).

2.4. Application à une poutre console encastrée

On considère la poutre encastrée constituée d'un matériau élastique parfaitement plastique soumise à une charge f donné Figure 2. La charge f est appliquée en 1 pas de charge. Les caractéristiques du matériau sont : module de Young $E = 210.0\ 10^9\ \mathrm{N/m^2}$ et contrainte limite $\sigma_y = 346.41\ 10^6\ \mathrm{N/m^2}$. Les résultats obtenus sont montrés Figure 2 et sont conformes à ceux obtenus en référence grâce au code de calcul CAST3M (Eyheramendy, 2004).

3. Décomposition de domaine en Java : une première approche

3.1. Stratégies parallèles en Java pour la décomposition de domaine

Les simulations en mécanique devenant de plus en plus importantes en terme de taille de problèmes, il est nécessaire d'utiliser des moyens de calcul de plus en plus puissants. La mise en commun de ressources hétérogènes peut également raisonnablement servir de base au calcul parallèle, et ce grâce au développement des capacités du réseau. On se trouve ainsi confronté à un double problème : avoir des codes pouvant tourner sur des architectures diverses pour un calcul parallèle, et avoir des codes qui supportent le renouvellement rapide des machines. L'environnement Java peut répondre à cette double problématique par ces capacité de portabilité et ces bibliothèques liées à la communication réseau. Dans notre approche, on s'appuie sur un concept de programmation à processus multiples basés la classe Thread de l'API Java (Flanagan, 2002).

3.2. Application à une méthodes de Schwarz recouvrante pour un modèle de Navier-Stokes incompressible

On s'intéresse à la formulation stabilisée des équations de Navier-Stokes en régime établi. Cette formulation est stabilisée est de type SUPG/PSPG (seuls le terme de convection et la pression sont pris en compte dans le terme de pondération

pour la stabilisation). On souhaite vérifier, d'une part, d'un point de vue numérique la robustesse de ce type de formulation (Schwarz-Newton-Krylov), et, d'autre part la robustesse informatique de la programmation multi-processus en Java. On résout le problème de la cavité entraînée pour environ 99000 ddls pour Re=3200 (voir Figure 3). Les calculs sont réalisés sur une machine Compaq ES45 (4 procs. 667 MHz, 2Go RAM).

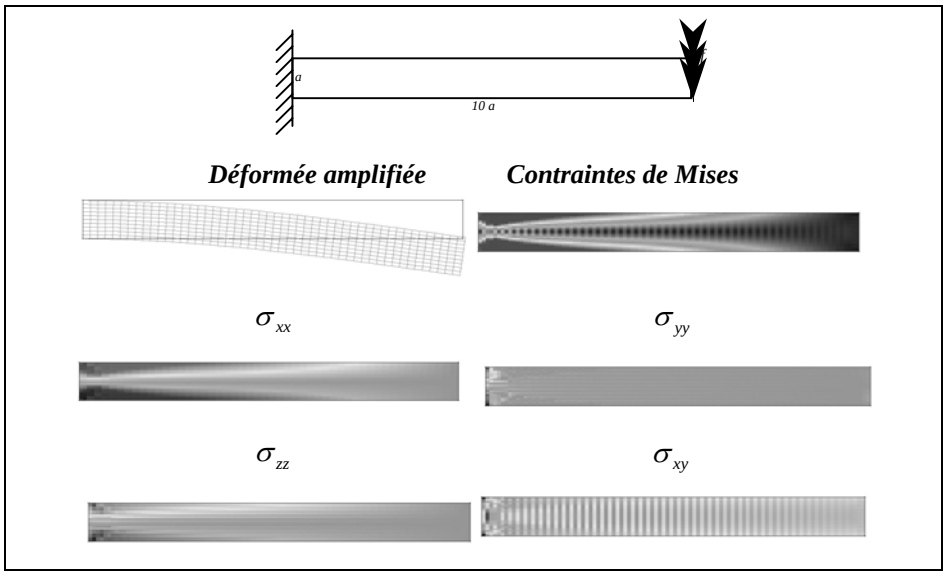
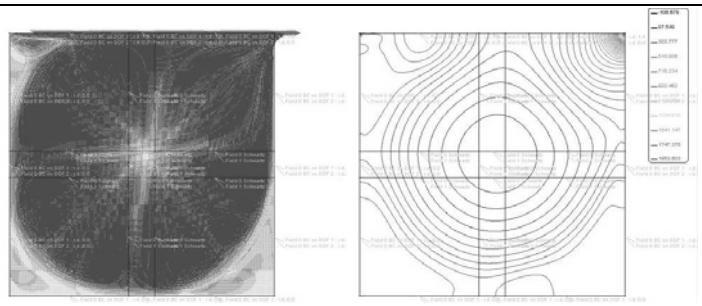


Figure 2. *Poutre console encastrée*

4. Conclusion

Dans cette communication, nous avons évoqué deux concepts de base d'une structuration avancée de code éléments finis en Java adaptée à la gestion de la complexité en mécanique. Le concept d'interface est présenté, et est illustré sur un exemple de formulation de plasticité parfaite J_2. On illustre d'autre part l'intérêt de s'appuyer sur un environnement permettant le développement de codes parallèles portables sur tout type de système. Ceci est illustré sur une méthode de décomposition de domaine avec éléments finis stabilisés pour le problème de Navier-Stokes de type Schwarz-Newton-Krylov. Ce travail tend principalement à montrer la faisabilité et l'intérêt d'une approche Java pour le calcul en mécanique. Cela ne représente que la première étape d'un travail qui consiste à développer des éléments de codes portables sur l'Internet pour réaliser des calculs en mécanique (voir http://vcmc.univ-lyon1.fr). L'avènement des réseaux informatiques laisse entrevoir des perspectives très intéressantes dans l'utilisation de systèmes très hétérogènes reliés par un réseau plus ou moins rapide. Le développement de codes

dits soit ‘généralistes’ soit ‘métiers’ est aujourd’hui moins crucial si l’on adopte des approches de type Java. Le choix d’outils de développement doit aussi être pris en compte dans les recherches futures. Aujourd’hui, il nous semble évident que les outils informatiques de type Java représentent vraiment la base convenable des environnements informatiques futurs en mécanique numérique.



Champ des vitesses ***Champ des pression***

Figure 3. *Cavité entraînée à Reynolds 3200.*

5. Bibliographie

L. Baduel, F. Baude, D. Caromel, C. Delbé, N. Gama, S. El Kasmi and S. Lanteri, « A parallel object-oriented application for 3-D electromagnetism », *ECCOMAS 2004*, Jyväskylä, Finland (2004).

D. Eyheramendy and D. Guibert, « A Java Approach for Finite Elements Computational Mechanics », *ECCOMAS 2004*, Jyvaskyla, Finland, July 2004.

D. Eyheramendy, « Le paradigme objet : une approche moderne pour gérer la complexité en mécanique », *Mémoire d’habilitation à diriger des recherches*, Université Lyon 1, 2004.

D. Flanagan, *Java in a Nutshell, Fourth edition*, Ed. O’reilly (2002)

R. Foerch, J. Besson, G. Cailletaud and P. Pilvin, « Polymorphic Constitutive Equations in Finite Element Codes », *Comput. Methods Appl. Mech. Engrg.*, 141, p. 355-372 (1997).

M. Ginsberg, J. Hauser, J. E. Moreira, R. Morgan, J. C. Parsons and T. J. Wielenga. « Panel session: future directions and challenges for Java implementations of numeric-intensive industrial applications », *Advances in Engineering Software*, vol. 31, 2000, p. 743-751.

N.T. Padial-Collins, W.B. VanderHeyden, D.Z. Zhang, E.D. Dendy and D. Livescu, « Parallel operation of CartaBlanca on shared and distributed memory computers », Concurency and Computation: Practice and Experience 16, p. 61-77 (2004).

J.C. Simo and T.J.R. Hughes, *Computational Inelasticity*, Interdisciplinary Applied Mathematics Serie n° 7, Springer, 1998.