Proceedings of the 8th WSEAS International Conference on APPLIED MATHEMATICS, Tenerife, Spain, December 16-18, 2005 (pp7-12)

An advanced object model for an operator split method Application to elastoplasticity

DOMINIQUE EYHERAMENDY
CDCSP / ISTIL – Institut Camille Jordan UMR 5208
Institut des Sciences et Techniques de l'Ingénieur de Lyon – Université Claude Bernard Lyon 1
15, Boulevard Latarjet, 69622 Villeurbanne Cedex
FRANCE
http://vcmc.univ-lyon1.fr

Abstract: -The object-oriented programming has been widely used in scientific computing since the 90s, including finite elements computations. The major consequence of it is a better maintainability and an easier extendibility of computational codes. In this paper, we advocate that the association of the natural high abstraction level of mathematics with an advanced object-oriented paradigm leads to a natural and a better organization of finite element codes. The approach is illustrated on an operator split method to solve an elastoplasticity problem. The implementation is done in Java.

Key-Words: - Finite elements, Operator split, Elastoplasticity, Object-oriented, Java.

1 Introduction

The object-oriented paradigm has been widely adopted in the finite element scientific computing community. The main reason for the strong interest for object-oriented technologies lies in the increasing size and complexity of the problems solved today. Most of the developments are today conducted under the C++ programming language. This approach has both advantages to support the object-oriented paradigm and to be efficient enough to deal with large scale computations. This is crucial from an industrial point of view. The practical interest of this kind of approach is that the researcher or the engineer can easily build a personal framework adapted to his problem: physical problem, numerical treatment, computational environment. This language has the major advantage to offer both a comfortable environment for object-oriented programming and a suitable numerical efficiency. Roughly speaking, the object-orientedness of the language tends to fasten the development and the maintenance of the codes, and the fact that the language is compiled ensures its performances. The development and the maintenance of code in C++ may lose interest for complex applications because of its complexity and to its strong dependence of the platform. Moreover, the language does not strictly enforce the object-oriented letting too many degrees of freedom to the programmer. We have shown in previous work that mathematical high level abstraction concepts developed in a single object concept leads to the design of new kind of tools in scientific computing (see e.g. [1] and [2]). The Java language is today used to implement the object-oriented paradigm in scientific computing (see [3] [4] and [5] for Java and e.g. [6] and [7] for references in C++).

Beyond the Java programming language adopted in the approach, we propose in the present paper, advanced approaches in object-oriented programming enhance the global architecture of finite elements codes. More natural and secure codes can be designed (see [8], [9] and [10]). In section 2, the elastoplasticity problem and the operator split method adopted in the solution scheme are briefly recalled. In section 3, we briefly describe a typical mechanism available in Java, called interface. This mechanism permits to enhance the consistency of finite element codes from a mathematical point of view. In section 3, the typical implementation is described on this example. A numerical example is given in section 4.

2 An Operator Split Method Applied To The Elastoplasticity Problem

2.1 Basic principles of operator split

Consider the following discrete equation where $x(t) \in \Re^N$ is the solution of the initial-boundary value problem with appropriate initial and boundary conditions:

$$\begin{cases} \dot{x}(t) = Ax(t) \\ x(t_n) = x_n \end{cases}$$

where $A : \Re^N \times \Re^N$ is a linear operator that can be split such as $A = A^1 + A^2$. The exact solution of the problem at time $t_{n+1} = t_n + h$ ($h > 0$ is the time step) is $x(t_{n+1}) = \exp((A^1 + A^2)h) \quad x_n$. This problem can be approximated by the following problem:

Problem 1 $\begin{cases} \dot{\bar{x}}(t) = A^1 x(t) \\ \bar{x}(t_n) = x_n \end{cases}$ Problem 2 $\begin{cases} \dot{x}(t) = A^2 x(t) \\ x(t_n) = \bar{x}(t_{n+1}) \end{cases}$

The solution of problem 1 is taken as initial condition for problem 2. The solution of this two-step problem is defined by a product formula: $x_{n+1} = \exp(A^1 h)\exp(A^2 h) \quad x_n$ where $\exp(A^1 h)$ and $\exp(A^2 h)$ are he exact solutions of problems 1 and 2. This is not the solution of the initial problem, unless A^1 and A^2 commute. This defines a first-order accurate algorithm (see [11] and [12]). We apply this solution scheme in a classical way to the solution of an elastoplasticity problem. The problem consists in finding an adequate operator split to obtain a converging stable algorithm.

2.2 Definition of the Elastoplasticity problem

We recall here the basic equations of the elastoplasticity in the case of perfect J_2 plasticity. The initial boundary value problem is summarized Fig. 1. The problem consists in finding the displacement field u and the stress field σ with appropriate regularity conditions such as defined in Fig. 1 and Fig. 2. In Fig. 2, the strain-stress relationship is summarized. An additive decomposition of strain is assumed. Perfect plasticity with an associated flow rule is considered. The yield condition is based on the second invariant of the deviatoric part of the stress field. More details about the model considered can be found in [12], [13].

2.3 Operator split applied to the elastoplasticity solution

We adopt a strategy of operator split to solve this problem. The problem summarized Fig. 1 and Fig. 2 is split, first, in a linear problem, the classical linear elasticity, and second, in a nonlinear problem corresponding to the constitutive law; the illustration is given Fig. 3. The solution of the global problem is obtained through a product-formula algorithm: the solution of the linear elasticity is solved with the initial conditions from the second problem at the previous time step. The stress state obtained such a way is not consistent. It does not satisfy the yielding condition. This trial stress state is then taken as initial conditions for the plastic correction problem. The elastic predictor which corresponds to the first part of the algorithm admits an exact solution and is reduced to a simple geometric update. The theoretical aspects of this product-formula problems go beyond the scope of this paper and can be found in e.g. [11] and [12].

Find $u(t,x)$ and $\sigma(t,x)$ with appropriate regularity conditions such that:

$$\sigma_{ij,j} + f_i = \rho\frac{\partial^2 u}{\partial t} \quad \text{on } \Omega\times T$$

$$\sigma_{ij} n_j = F_i \quad \text{on } \partial_2\Omega\times T$$

$$u_i = \bar{u}_i \quad \text{on } \partial_1\Omega\times T$$

$$u(0,x) = u_0(x) \quad \text{on } \Omega$$

$$\varepsilon_{kl}(u) = \nabla^S(u) = \frac{1}{2}(u_{k,l} + u_{l,k}) \quad \text{on } \Omega\times T$$

+ Consitutive law

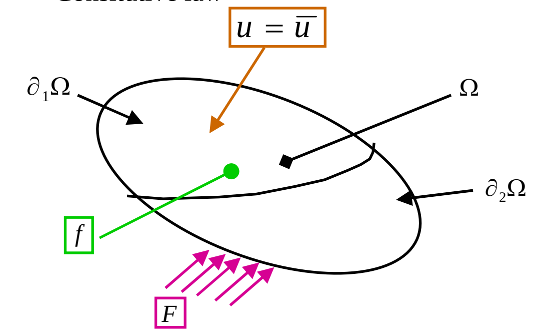


Fig. 1 Initial boundary-value problem for elastoplasticity

Perfect elastoplasticity (associated flow rule, Mises criterion)

— Additive decomposition of the strain tensor $\dot{\varepsilon} = \dot{\varepsilon}^e + \dot{\varepsilon}^p$

— Yield condition : $f(\sigma) = J_2(\sigma) - R$

— Flow rule $\dot{\varepsilon}^p = \gamma\frac{\partial f}{\partial\sigma}$

$\gamma \geq 0$

— With conditions $f(\sigma) \leq 0$

$\gamma f(\sigma) = 0$

Where $R = \sqrt{\frac{2}{3}}\sigma_Y$ (σ_Y is the yield criterion)

Where J_2 is the second invariant of the deviatoric part of σ.

Fig. 2 Elastoplasticity constitutive model

Elastoplastic		**Elastic predictor**		**Plastic corrector**
$\dot{\varepsilon} = \nabla^S(\Delta\dot{u})$ $\dot{\varepsilon}^p = \gamma \frac{\partial f}{\partial \sigma}$	=	$\dot{\varepsilon} = \nabla^S(\Delta\dot{u})$ $\dot{\varepsilon}^p = 0$	+	$\dot{\varepsilon} = 0$ $\dot{\varepsilon}^p = \gamma \frac{\partial f}{\partial \sigma}$

Fig. 3 Operator split method applied to elastoplastcity

3 A mathematical based advanced object-oriented implementation

3.1 Advanced object-oriented techniques

A description of basic concepts of object-oriented mechanism in the context of finite elements can be found in [14], [15] and [8] and references therein. In addition to the basic concepts, the Java language introduces new concepts such as the concept of inner class and the concept of interface. These new features permit the numerician to build safer and more robust computational applications, typically for complex finite elements applications. These mechanisms are thoroughly described and illustrated in [8], [9] and [10]. In this paper, we wish to focus on the interface mechanism. An interface is a reference type that is closely related to a class. It can be seen as a pure abstract class (roughly speaking a class that has no instance). It does not define any implementation but only specifications, i.e. only methods that are defined to be mandatory in a given class. A class is said to implement an interface, if and only if, the class exhibits an implementation of the methods specified at the level of the interface. As matter of fact, a class can inherit methods from a superclass, and must implement a set of methods imposed by the implementation of an interface. A complete description of the interface mechanism can be found in [17], and applications in the context of finite elements codes in [8] and [9].

3.2 Algorithmic consistency enforcement in finite elements: Application to elastoplasticity

We apply this mechanism to the plastic corrector phase (Problem 2 in the operator split algorithm Fig. 3). The global framework will be detailed in a forthcoming paper [8]. This problem can be solved here using a classical return algorithm. This algorithm can be viewed as a backward Euler scheme. The algorithm is given under an incremental form in Fig. 4. The principle of the algorithm is to consider the elastic predictor step. If the stress state is outside the yield surface, the plastic correction is performed. It is worth noting that the algorithm can be performed locally overall the domain. From a practical point of view, in the context of finite elements applications, the correction step is classically performed at the level of gauss points (numerical integration points). More details about the algorithm can be found e.g. in [12] or in [16] for more technical details. It is obvious that this algorithm cannot be applied to any equation constitutive models. It is a restriction of a general return-mapping algorithm for the J$_2$ plasticity. The programmer is in charge to maintain this consistency between the physical model, the numerical model and the implementation. The interface mechanism offers the programmer to enforce this global consistency at the level of the code. Thus, mathematical properties are implemented in a natural way, providing robustness through mathematical foundations.

Problem at iteration i and step n+1 :

•**Given** σ_n **and** $d\varepsilon_{n+1} = B\,\Delta d^i$ **, find** σ_{n+1}

(

–***Compute trial stress (from elastic predictor problem)***

$$\sigma_{n+1}^{tr} = \sigma_n + d\sigma^{tr} = \sigma_n + D^{el} d\varepsilon_{n+1}$$

–***If*** $f(\sigma_{n+1}^{tr}) \le 0$ ***then*** $\sigma_{n+1} = \sigma_{n+1}^{tr}$ ***and stop***

–***Else compute plastic correction***

$$d\sigma^p = -D^{el} d\gamma \frac{dq}{d\sigma}$$

Where the plastic multiplier is

$$d\gamma = \frac{f(\sigma_{n+1}^{tr})}{\frac{df}{d\sigma}^T D^{el} \frac{dq}{d\sigma}}$$

Fig. 4 Radial return algorithm for J_2 elastoplasticity

2.1 Principles of the implementation

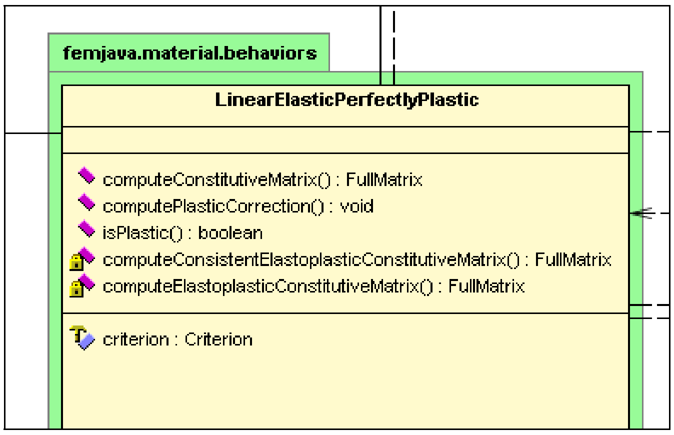
In this section, we pose the basic principles of a mechanism to enforce mathematical consistency in scientific computing applications. This is illustrated here in the context of finite elements, on the example of numerical integration of constitutive equations in a two-step operator split algorithm. The object model is given in Fig. 5. In this example, the constitutive equations are represented by a generic class called **Behavior**. This class is the abstract class representing all the different types of constitutive laws. In this example, the subclass **LinearElasticPerfectlyPlastic** represents the typical constitutive model studied here: linear elasticity – perfect plasticity (yield criterion based on the J_2 function). This class provides all the behavior needed to compute the components corresponding to this typical model. The abstract class **Integrator** represents all the generic behavior of the different types of integrators, i.e. roughly speaking a single generic method called ***integrate***. This method initiates the correction phase. The subclass **RadialReturnAlgorithm** strictly implements the algorithm posted Fig. 4. The latter partially describes a classical object-oriented model for constitutive law modeling. Here, we propose to enforce the integrability of the model of equations through the interface described Fig. 6. The interface **Integrable** specifies the methods needed by all the models of integrators: the computation of the constitutive matrix and the determination of the plastic condition (checking of the yield condition). The subinterface **RadialReturnAlgorithm** specifies the methods needed in the algorithm Fig. 4, i.e. the computation of the plastic correction. The last step is to define the class **LinearElasticPerfectlyPlastic**, subclass of class **Behavior**, to implement the interface **RadialReturnIntegrable**. The class **LinearElasticPerfectlyPlastic** implements the methods specified in both, the interface **Integrable** and the interface **RadialReturnIntegrable**. The programmer is in charge of the correct use of the numerical algorithm in the context of the model of equation. The class **LinearElasticPerfectlyPlastic** is given Fig. 7. The methods implementing the interface are: ***isPlastic*** (checking of the yield condition), ***computePlasticCorrection*** (computation of the plastic correction) and ***computeConstitutiveMatrix*** (computation of the constitutive matrix). A further extension of the code to implement a new set of constitutive equations is possible, if and only if, the programmer specifies and implements the solution scheme for the correction step of the global algorithm. The consistency of the code is then guaranteed.

- **Object**
  - **Behavior**
    - **LinearElasticPerfectlyPlastic**
  - **Integrator**
    - **RadialReturnAlgorithm**

Fig. 5 Partial view of the class hierarchy for the constitutive equations description

- **Integrable**
  - **RadialReturnIntegrable**

Fig. 6 Hierarchical organization of interfaces for numerical integration



femjava.material.behaviors

LinearElasticPerfectlyPlastic

computeConstitutiveMatrix() : FullMatrix
computePlasticCorrection() : void
isPlastic() : boolean
computeConsistentElastoplasticConstitutiveMatrix() : FullMatrix
computeElastoplasticConstitutiveMatrix() : FullMatrix

criterion : Criterion

Fig. 7 Detail of the class for J_2 plasticity equations

4 Numerical example: a clamped beam

The code is tested on the example of a clamped beam submitted to a vertical load. The beam posted Fig. 8 is loaded with *f*. The load *f* is applied in one step. The characteristic of the material are: the Young modulus $E = 210.0\ 10^9\ \text{N/m}^2$ and the yield criterion $\sigma_y = 346.41\ 10^6\ \text{N/m}^2$. The results that are shown in Fig. 9 are in good qualitative and quantitative agreement with the one obtained with the code CAST3M (see [18]): stresses are similar and the vertical displacement of the bottom left corner is 2.18 10^{-1} m. Plane strains hypothesis are considered in this example.

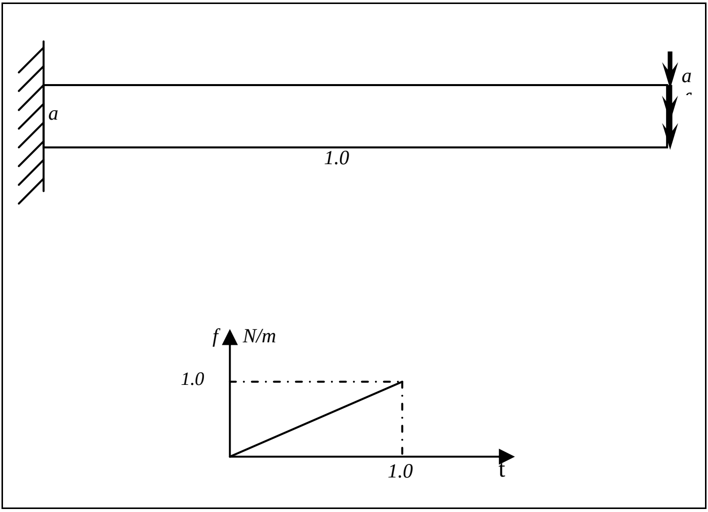


Fig. 8 Clamped beam

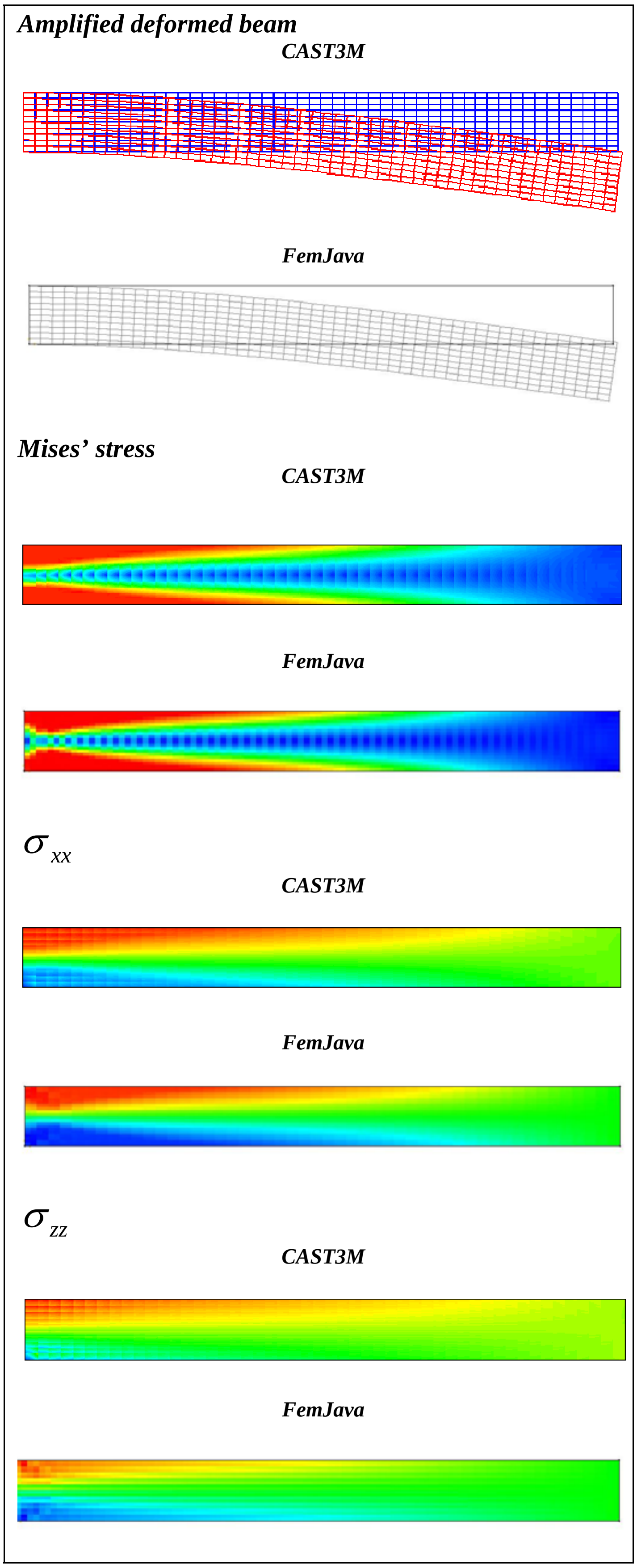


Fig. 9 Numerical application to a clamped beam

4 Conclusion

In this paper, we have presented a mechanism based an advanced object-oriented model to enforce the mathematical consistency of computational applications. The concept of interface provided by the Java language represents the convenient framework to design more robust and safer finite element codes. The approach has been illustrated on an elastoplasticity problem. An elastic-plastic operator split method is proposed as framework for the solution. We have focused in this example on the plastic correction step for which a numerical algorithm is mandatory to compute a consistent stress state. In the design proposed here, the implementation of the physical model (J_2 elastoplasticity) is clearly separated from the implementation of the numerical algorithm (radial return). The relationship is enforced by the interface mechanism implemented at the level of the set of equations defining the constitutive law. The algorithm has a generic implementation. The behavior, embedding the equations, implements the methods to compute the specific contributions. The mechanism of interface permits the programmer to enforce the consistency between both, the behavior and the algorithm. To go further, the Java language offers additional advanced object-oriented features, such as inner classes. Similarly, these mechanisms significantly enrich the abstraction capabilities of classical object-oriented approaches. Beyond the use of a the Java language, we advocate that high level abstraction programming concepts will have a growing interest in a near future to design the next generation of safe and reliable computational applications. In [1] and [2], the variational formulation derivation tools and finite elements symbolic forms associated to an automatic programming schemes has shown the possibility to introduce high level mathematical concepts in the design of finite elements codes. High level control features can be extended at the symbolic level. These ideas could merely be extended to the management of complex simulations in the broad domain of verification and validation of code. The integration of symbolic concepts directly embedded in the numerical simulation tools will permit to enhance the reliability of computational tools. Advanced oriented-object concepts represent a key ingredient to achieve high level abstraction simulation tools.

References:

[1] D. Eyheramendy, An object-oriented hybrid Symbolic/Numerical Approach for the Development of Finite Element Codes, *Finite Element Analysis and Design*, Vol. 36, 2000, pp. 315-334.

[2] D. Eyheramendy and Th. Zimmermann, Object-oriented finite elements : IV. Application of symbolic

derivations and automatic programming to nonlinear formulations, *Computer Methods in Applied Mechanics and Engineering*, vol. 190 n° 22-23, 2001, pp. 2729-2751.

[3] 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*, Vol. 16, p. 61-77 (2004).

[4] 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).

[5] D. Eyheramendy, Object-oriented parallel CFD with JAVA, *15th International Conference on Parallel Computational Fluid Dynamics*, Eds. Chetverushkin, Ecer, Satofuka, Périaux, Fox, Ed. Elsevier, 2003, pp. 409-416.

[6] F. Hecht, FREEFEM++, A finite element PDE solver, *ECCOMAS 2004*, Jyväskylä, Finland (2004).

[7] Y. Dubois-Pèlerin and Th. Zimmermann, Object-oriented finite element programming : III. An efficient implementation in C++, *Comput. Methods Appl. Mech. Engrg.*, Vol. 108, 1993, pp. 165-183.

[8] D. Eyheramendy**,** Object-Oriented Finite Elements programming in JAVA: I - Basic Principles, *Preprint CDCSP 04-01 In preparation*, CDCSP Lyon 1 University, 2005.

[9] D. Eyheramendy, Object-Oriented Finite Elements programming in JAVA: II - Implementation, *Preprint CDCSP 05-02 In preparation*, CDCSP Lyon 1 University, 2005.

[10] D. Eyheramendy**,** Object-Oriented Finite Elements programming in JAVA: III - Applications, *Preprint CDCSP 05-03 In preparation*, CDCSP Lyon 1 University, 2005.

[11] A. Chorin, T.J.R Hughes, M.F. McCracken and J.E. Marsden, Product formulas and Numerical Algorithms, *Communications on Pure and Applied Mathematics*, No.31, 1978, pp. 205-256.

[12] J.C. Simo and T.J.R. Hughes, *Computational Inelasticity*, Springer, 2000.

[13] Lemaitre and J.L. Chaboche, *Mécanique des matériaux solides*, Dunod, 1996.

[14] Y. Dubois-Pèlerin, Th. Zimmermann and P. Bomme, Object-oriented finite element programming : II. A prototype program in Smalltalk, *Comput. Methods Appl. Mech. Engrg.*, Vol. 98, 1992, pp. 361-397.

[15] Th. Zimmermann, Y. Dubois-Pèlerin and P. Bomme, Object-oriented finite element programming : I. Governing principles, *Comput. Methods Appl. Mech. Engrg.*, Vol. 98, 1992, pp. 291-303.

[16] S. Commend and Th. Zimmerman, Object-Oriented Nonlinear Finite Element Programming : a Primer, *Adv. In Soft. Engnrg*, 32-8, 2001, 611-628.

[17] Flanagan, *Java in a Nutshell, Fourth edition*, Ed. O'reilly (2002)

[18] P. Verpeaux, T. Charras et A. Millard, CASTEM 2000 : une approche moderne du calcul de structures, *Calcul des structures et intelligence artificielle*, Vol. 2, J.M. Fouet, P. Ladevèze et R. Oyahon Eds., Pluralis, 1988.