New tracks for future computational platforms for engineering applications

Dominique Eyheramendy

New tracks for future computational platforms for engineering applications

D. Eyheramendy
CDCSP/ICJ UMR 5208, Institut des Sciences et Techniques de l'Ingénieur de Lyon,
Université Claude Bernard Lyon 1
15, Blvd Lartarjet , FR-69622 Villeurbanne Cedex

Keywords: Object-oriented programming, Finite Elements, Web applications, Java, Darcy's flow, J_2 plasticity.

This purpose of this paper is to address new tracks for the future generation of computational applications in mechanics and related branches. We advocate that modern computational tools will have to deal with complex strongly coupled multi-physics multi-scale problems. Moreover, heterogeneous distributed multi-processors systems are used today for the numerical simulations. We pose here some basic ideas for the design of modern computational applications. All the illustrations are based on finite elements strategies implemented in a pure Java paradigm.

1 Introduction

Nowadays, the engineers and the researchers have to take into account a large amount of parameters in the design process of computational applications: efficiency, need of high level concepts for fast prototyping, scale of computations, need of multi-processing computation, networking, need of high level GUI (graphic user interface),… In this context, the strong interest for object-oriented technologies in computational mechanics lies in the increasing size and complexity of the problems currently solved (see Noor [1] for example in computational structures mechanics). This approach has been investigated in many computational fields in mechanics: constitutive law modelling, in finite deformation plasticity, in parallel finite element applications, in rapid explicit dynamics, in fracture mechanics (see Eyheramendy [3] and [4] and references therein). The broad range of applications solved within the object-oriented paradigm shows that every researcher or engineer can easily build a personal framework adapted to his domain of interest: physical problem, numerical treatment, computational environment... The challenging problems for the years to come will certainly concern coupled multi-scale multi-physics. Moreover, the increasing size of the problems will lead to the development of efficient parallel strategies. From a technical point of view, the developers need to

mix multiple software technologies (graphical libraries -e.g. OpenGL-, communications libraries -e.g. MPI-...) in order to integrate the numerical algorithms into computer tools. Considering the inherent complexity of the traditional languages such as C++, the development of global frameworks for finite elements computations can rapidly become cumbersome. The software strategy we proposed is based on a pure Java object-oriented paradigm. This platform has the major advantage to offer both a comfortable environment for object-oriented programming and a suitable numerical efficiency. Moreover, from an industrial point of view, it is worth to notice that the code naturally becomes platform independent. We advocate that the C++ language could merely be replaced by a Java type approach. This has already be advocated by many authors: e.g. Ginsberg & al. [5], Padial-Coolins & al. [6], Baudel & al. [7], Eyheramendy [8], Bull & al. [9], Nikishkov & al. [10], Häuser & al. [11], Riley & al.[12].

In section 2, two advanced object-oriented concepts are briefly presented. The objective of this work is to closely relate the developments to the physical and mathematical modeling. This is illustrated, first in section 3, on the enforcement of global model consistency on the example of Darcy's flow formulations, and second in section 4, on a numerical model for the mathematical consistency enforcement applied to the elastoplasticity. At last, networking strategies and an Internet integration are briefly discussed in section 5. An example of web portability is presented.

2 Advanced O.O concepts in Java

Basic object-oriented concepts and applications to finite elements has been widely discussed in Zimmerman & al. [1] and Dubois-Pèlerin & al. [13] and [14] and related pioneering works. The first concept introduced in the present work is the concept of inner class. The definition of a class is allowed within another one. Both classes partially share some data. We used this scheme to define at the same local level, first, data defining the fields and the finite elements formulations overall the computational domain, and second, data defining the fields and the formulations at the elemental level. The second concept is the concept of interface. An interface is a reference type that is closely related to a class. It can be seen as a pure abstract class (a class that cannot be instanced). This class 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. The complete description of these concepts goes beyond the scope of this paper and can be found in Eyheramendy [15] or Flanagan [16].

3 Local and global data code consistency: Application to Darcy's flow

We give an example for which special Java syntaxes may be helpful to better handle complexity of a F.E code. Inner classes allow the programmer to partially hide information to the whole of the code. It can be interesting to partially share global and local aspects of a numerical scheme. The approach is illustrated on a

simple interface tracking scheme for underground water flow and a mixed formulation of the Darcy's flow equations. The management of the finite element formulation is described on this example.

3.1 A free surface seepage problem

3.1.1 Seepage problem formulation: strong and weak forms

A procedure to locate the free surface of an unconfined seepage flow through porous media is completely described in Lacy [17]. Let us briefly recall the problem. We consider the flow of an incompressible and homogeneous fluid into a porous medium. The medium is assumed to be either wet (saturated) or dry. Capillarity, partial saturation and evaporation are neglected. The free surface is defined as the boundary line between the dry and wet soils as shown in figure 1 (free surface CDE). No flux gets through the free surface and the pressure is zero on the free surface. The domain Ω_w represents the flow region (the saturated part of the geometric Ω domain occupied by the earth structure) and Ω_d the dry part.

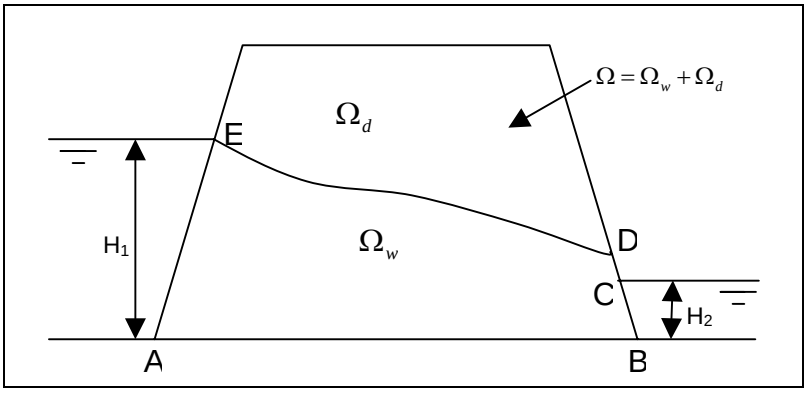


Figure 1 - Definition of the free seepage problem.

The piezometric head Φ is defined with respect to the pressure p such as:

$$\phi = \frac{p}{\rho g} + y = \frac{p}{\gamma} + y$$

where ρ is the density of the fluid and γ the specific weight of the fluid.

The steady state problem may be modeled as follows:

$p \geq 0$ in Ω_w

$p = 0$ in Ω_d

$\nabla . k . \nabla \phi = 0$ in Ω

k: Darcy permeability tensor.

With boundary conditions:

$p = \gamma(H_1 - y)$ on AF

$p = \gamma(H_2 - y)$ on BC

$p = 0$ and $n . k . \nabla \phi = 0$ on ED

$p = 0$ and $n . k . \nabla \phi = 0$ on CD

$n . k . \nabla \Phi = 0$ on AB

The fundamental difficulty is that the location of the free surface is unknown a priori and the enforcement of the boundary conditions on the free surface may be tremendous. The formulation is extended to the entire domain $\Omega = \Omega_w \cup \Omega_d$ by using an extended pressure field *p(x)* for which the $p(x) = 0$ in Ω_d -the dry soil domain-. Existence and uniqueness of the solution for the extended problem is showed by introducing the following penalized problem:

$$\begin{cases} \text{Find } p_\varepsilon \in P = H^1(\Omega_w) \text{ such that } \forall q \in H_0^1(\Omega_w) \\ \int_\Omega \frac{1}{\gamma} k \nabla p_\varepsilon \nabla q d\Omega = -\int_\Omega k\, H_\varepsilon(p) \nabla y \nabla q d\Omega \end{cases}$$

$H_\varepsilon(p)$ is an extension of the Heaviside function that includes a penalty parameter. Since the location of the free surface is unknown a priori, a Newton iterative scheme is required to solve the problem. A local computation (at the elemental level) is done to determine of the penalty parameter ε and a cut-off pressure p_o which enforce the pressure to a "small" negative value in the dry soil. A detailed description of it can be found in Lacy [17].

3.1.2 Implementation in the Java code

The main object in the application is the field. It supports all the nodes and the values of degrees of freedom. The formulation initializes the fields. The formulation for the seepage problem is based on the definition of a single scalar field, the pressure.

The implementation of the formulation is made through a class called: **PressureDarcyPenalizedProblem**. The class is posted in Figure 2. The class subclasses **Formulation** in which all the basic behavior of the finite elements formulation is taken into account, e.g. the global way of building fields overall the computational domain. The class embeds an inner class called **PressureDarcy**. The two important methods in the class are:

- *initialize()* which permits us to describe the unknown field, here a scalar field, the pressure, for the problem; this a global method (see figure 3),
- *getElement()* which permits us to instantiate at the local level the finite element called **PressureDarcy** with local elemental fields, numerical quadrature (which may play a crucial role for constitutive law modeling) and material definition.

The finite element **PressureDarcy** exhibits as local behavior all the computation of the finite element matrices defined Figure 2 in italic and Figure 3. The definition of both classes, i.e. the inner and outer classes, permits us to completely define a new finite element formulation. This special feature of the Java programming language can be seen as a generalization of the object-oriented concept at the level of a class. This leads to an enhanced organization of code.

```
public class PressureDarcyPenalizedFormulation extends Formulation
{
   public static double Tolerance = 1e-3 ;
   public static double LargeParameter = 1e10 ;

   public String toString()  { // … }
   public Material defaultMaterial()  { return new PorousMedia () ; }
   public void initialize( Domain domain ) { // … }

   public Element getElement( ElementalGeometry aGeom , Quadrature aQuadrature ,
                              ElementalField[] flds, int nb , Material m )
   {
      return new PressureDarcy ( aGeom , aQuadrature , flds , nb , m ) ;
   }

   public static class PressureDarcy extends Element
   {
      public PressureDarcy( ElementalGeometry aGeom , Quadrature aQuadrature ,
                            ElementalField[] flds , int n , Material m ) {  // … }
      public FullMatrix computeConstitutiveMatrix() {  // … }
      public Hashtable computeElementalMatrices( TimeStep ts ) { // …}
      protected double computeCutOffPressure( ElementalGeometry aGeometry ,
                            double gamma ) { // …}
      protected double computeEpsilon( ElementalGeometry aGeometry , double gamma ) { // …}
      // … additional non-implemented abstract methods …
   }
}
```

Figure 2 - Implementation of an inner class in Java for defining a formulation

```
public void initialize( Domain domain )
{
      Field[] fields = new Field[1] ;
      fields[0] = domain.createAScalarField ( 0 ) ;  // PRESSURE
      domain.setFields ( fields ) ;

      domain.setNumberOfUnknownFields ( 1 ) ;

      Subdomain[] subdomains = domain.getSubdomains () ;
      for( int i = 0 ; i < subdomains.length ; i++ )
         this.initialize ( subdomains[i] ) ;
}

public Element getElement( ElementalGeometry aGeom , Quadrature
            aQuadrature , ElementalField[] flds , int nb , Material m )
{
      return new PressureDarcy ( geom , gaussPoints , fields , nb , m ) ;
}
```

Figure 3 - Definition of the fields the Darcy's flow formulation

3.1.3 Numerical examples

For all the test cases, the permeability k is set to identity tensor and the specific weight of water γ is set to 1.0. Bilinear quadrilateral elements have been used. We obtain similar results to the one obtained in Lacy [17]. The first example in figure 4 shows a dam with a toe drain, and the second one, a dam with an impermeable sheet on the upper part of the right face. We tested on these cases various strategies for the choice of the penalty parameter ε and cut-off pressure definition. The accuracy of the solution drastically depends on the choice of various parameters involved in the formulation (penalty parameter, cutoff pressure…). The way to impose the boundary conditions for the pressure especially on the free surface seepage zone may lead to instable results. These experiments lead us to adopt an alternative strategy for the solution or free surface seepage problem by introducing a free surface equation and solving the flow using a velocity-pressure formulation. In the following section a mixed velocity-pressure is studied.

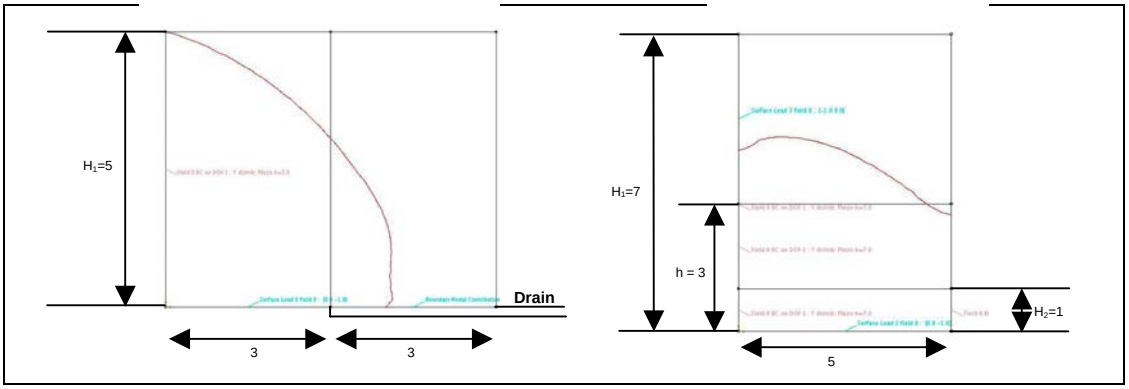


Figure 4: Dams: toe drain and impermeable sheet on the downstream face.

3.2 A mixed stabilized formulation for Darcy's flow

3.2.1 Mixed stabilized formulation

We consider a pressure-velocity formulation for a Darcy' flow problem. The free surface tracking is not considered here; we aim to get a stable and accurate formulation for a simple Darcy's flow. A mixed formulation is adopted. The problem can be summarized as follows:

Find u velocity and p pressure with appropriate regularity conditions such that (domain described Figure 5):

$$u = -k\nabla\left(\frac{p}{\gamma} + y\right) \quad \text{on } \Omega$$

$$div(u) = 0 \quad \text{on } \Omega$$

$$u \cdot n = 0 \quad \text{on } \partial\Omega$$

where:

ρ	fluid density
g	gravity
$\gamma = \rho g$	charact. fluid heigth

The variational formulation is stated as follows (with the definition of appropriate regular spaces):

Find $(u, p) \in V \times P$ such that $\forall (v, q) \in V \times P$:

$$\int_{\Omega} v\, k^{-1} u \, d\Omega - \int_{\Omega} div(v)\; p \, d\Omega + \int_{\Omega} q \; div(u) \, d\Omega$$

$$+ \frac{1}{2} \sum_{\Omega^e} \int_{\Omega^e} (\gamma\, k^{-1}\, u + \nabla p + \nabla \gamma y) \frac{k}{\gamma} (\gamma\, k^{-1}\, v - \nabla q) \, d\Omega \quad = \quad \int_{\Omega} div(v)\, \gamma y \, d\Omega$$

A more detailed description of the formulation proposed by Masud & al. [19] can be found in Eyheramendy & al. [18].

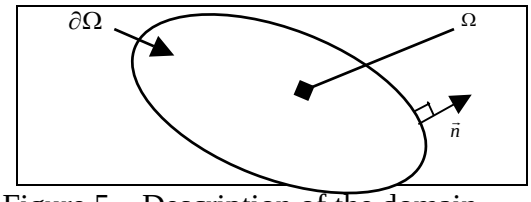


Figure 5 – Description of the domain

3.2.2 Implementation

The implementation is similar to the one presented in the previous example. The new formulation is introduced through a new set of classes. This is done in the same way as in the previous example. This formulation has two unknowns: the pressure field and the velocity field. Their natural definition of the fields at the global and local levels is exhibited in the figure 6. It can be compared to the one of figure 3. We show on this example the full potential of the inner class concept to achieve a natural definition of the unknowns of the physical model and to enforce the consistency of the finite element model at the elemental level within the application.

```
public void initialize( Domain domain )
{
      Field[] fields = new Field[2] ;

      fields[0] = domain.createAVectorField ( 0 ) ;  // VELOCITY
      fields[1] = domain.createAScalarField ( 1 ) ;  // PRESSURE
      domain.setFields ( fields ) ;

      domain.setNumberOfUnknownFields ( 2 ) ;

      Subdomain[] subdomains = domain.getSubdomains () ;
      for( int i = 0 ; i < subdomains.length ; i++ )
         this.initialize ( subdomains[i] ) ;
}
```

Figure 6 – Definition of the velocity and pressure fields

3.2.3 Numerical results

We study in this example the injection of a fluid at the corner of a square domain. Similar physical parameters as in the previous formulation are used in this simulation. The results are in good agreement with the ones of Masud & al. [19].

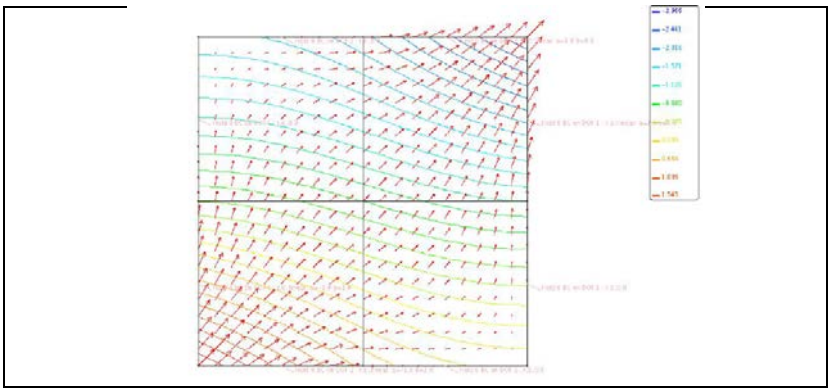


Figure 7 – Injection at a corner for a Darcy's flow

4 Nonlinear material modeling: enhanced data organization to enforce numerical consistency

4.1 Elastoplasticity – Radial return algorithm

We recall here the basic equations of the elastoplasticity in the case of perfect J_2 plasticity. Classical notation are adopted. The problem consists in finding the displacement field u and the stress field σ with appropriate regularity conditions such as defined in figure 8. 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. The solution of the global problem is obtained through an operator split technique (see figure 9): a trial solution of the linear elasticity is solved with the initial conditions from the second problem at the previous time step. More details about the problem definition and the solution scheme can be found in Commend & al. [22], Simo & al. [20] or Lemaître & al. [21].

4.2 An advanced object implementation: Algorithmic consistency enforcement in finite elements: Application to elastoplasticity

We apply the mechanism of interface to the implementation of the plastic corrector phase, for the numerical integration of the constitutive law. A classical return algorithm is used. The global framework will be detailed in a forthcoming paper Eyheramendy [23]. 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). This algorithm cannot be applied to any constitutive model. It is a restriction of a general return-mapping algorithm for the J_2 plasticity. The algorithm is given figure 10.

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

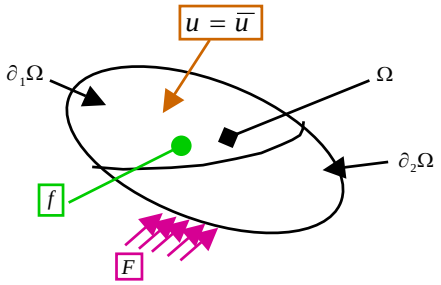
$$\sigma_{ij,j} + f_i = \rho \frac{\partial^2 u}{\partial t} \quad \text{on } \Omega \times T$$

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

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

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

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



Constitutive Law:

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

Flow rule $\dot{\varepsilon}^p = \gamma \frac{\partial f}{\partial \sigma}$ with conditions $\gamma \geq 0$, $f(\sigma) \leq 0$, $\gamma f(\sigma) = 0$

Figure 8 - Elastoplasticity constitutive model

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

Figure 9 - Global solution scheme for elastoplasticity

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)***

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

  - ***If*** $f(\sigma^{tr}_{n+1}) \leq 0$ ***then*** $\sigma_{n+1} = \sigma^{tr}_{n+1}$ ***and stop***
  - ***Else compute plastic correction***

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

Figure 10 - Radial return algorithm

Without entering the details of the class hierarchy, a partial view of the object model is given in figure 11. 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 J_2 plasticity. 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 figure 10. 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 figure 10, 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 figure 12. 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). The consistency of the code is then guaranteed. The programmer is responsible for maintaining this consistency between the physical model, the numerical model and the implementation. The interface mechanism permits 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.

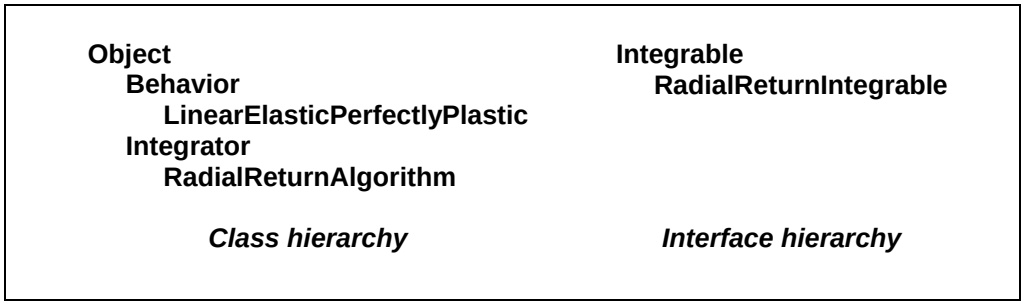


Figure 11 - Partial view of the class hierarchy for the constitutive law integration

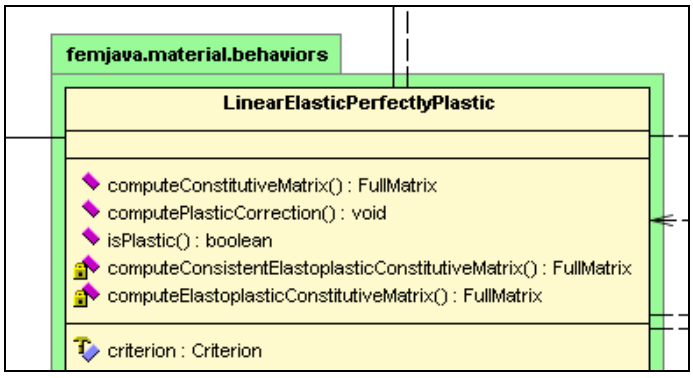


Figure 12 - Detail of the class for J_2 plasticity equations

4.3 Numerical example: a strip footing problem

We briefly present here a problem of the bearing capacity of a surface footing. The data describing the problem (domain and physical data) is given in figure 13. The failure load obtained in this example is $q_f = 5.2\ kN/m$ (see figure 14). This value is in good agreement to the approximated theoretical value $q = 5\ kN/m$ (see Commend & al. [22]). A detail description of the problem and the typical numerical ingredients used in this simulation will be discussed in the forthcoming paper Eyheramendy [23].

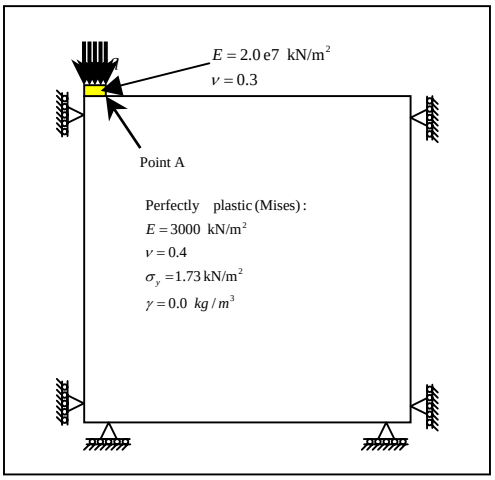


Figure 13 – Definition of the strip footing problem

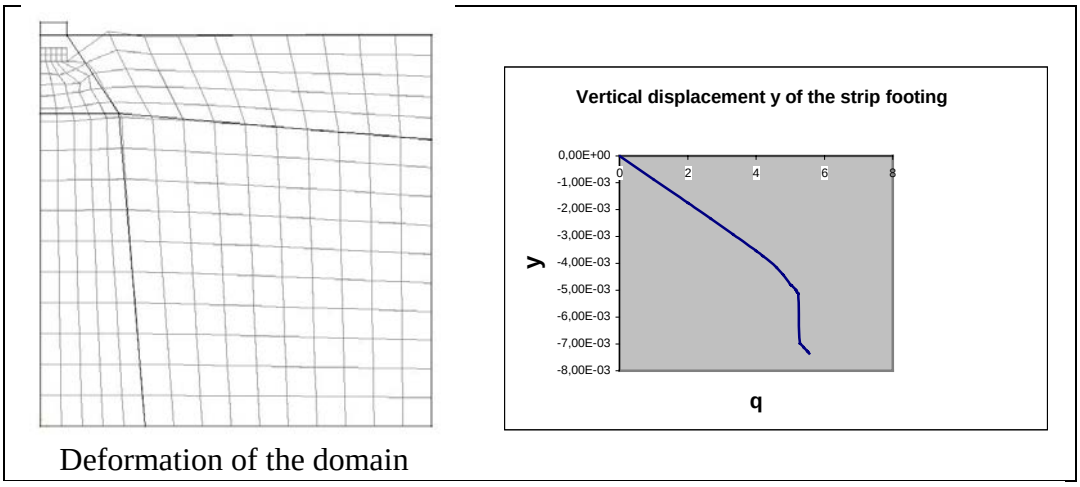


Figure 14 - Numerical results for the strip footing problem

5 High level portable libraries and integration of F.E over the Internet

One problem today requiring attention for engineering computational applications is the use of the cyber infrastructure. In modern computational mechanics, a pure computer science approach cannot be able to carry out new designs in finite elements codes. Knowledge and expertise from the mechanical and mathematical modeling has to be integrated into complex applications to be run on a complex system of computers. We describe here a simple example of distribution of a finite elements application in mechanics over the Internet. It will probably be for the near future a hot topic for the distribution of commercial applications. A new way to use computational tools is to be invented. The major point of the following lies in the portability of the code. This is a typical feature of an application developed using a pure Java paradigm. In figure 15, an example of WWW integration of computational application is presented. The principle lies in the fact that every Internet browser in the world embeds a Java Virtual Machine which allows the execution of Java code on the local computer. The computational mechanics application is made accessible through a simple HTML page by the way of a typical Java framework called Applet (see Flanagan [16]). A completely portable GUI (Graphical User Interface) is offered to the Web user. A convenient user interface gives access to the specific application. In the example shown here, typical problems of computational mechanics are available: strip footing (J_2 plasticity), various cavity flows (Stokes, Navier-Stokes), flow around a cylinder for an incompressible Navier-Stokes flow… This development is an example of distribution of an application over various computer systems.

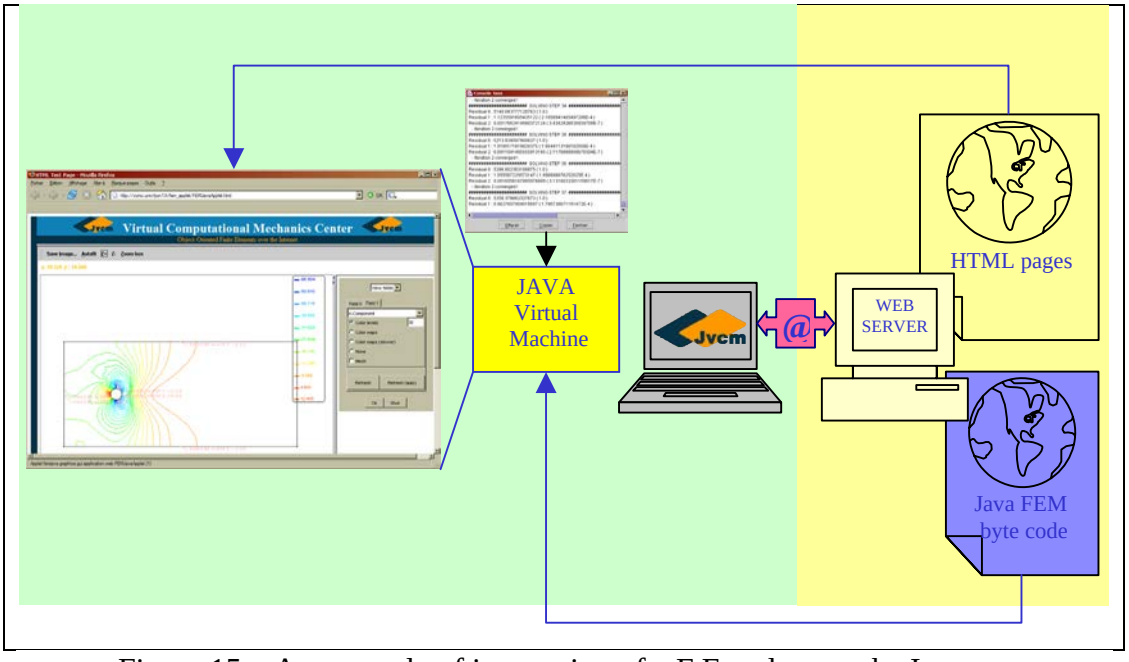


Figure 15 – An example of integration of a F.E code over the Internet

6 Conclusion

In this paper, we have presented basic principles to enhance the consistency of finite elements code for engineering applications. First, an example of local and global consistency enforcement for a finite element model has been described and illustrated on different Darcy's flow finite elements formulations. Then, the typical interface mechanism applied to the numerical consistency enforcement has been described. Both mechanisms help the programmer to produce safer and better code based on some mathematical properties. At last, general ideas about the integration of computational application over the Internet have been discussed and a basic application briefly described.

We advocate that computational frameworks will naturally bend on high abstraction mathematical concepts. This opens new tracks in the crucial domain of verification and validation of code by naturally embedding mathematical aspects into the computational codes. The second aspect of the paper is the need of high level integration of computational applications including the use of the cyber infrastructure.

The portability of computational codes based on high level complex libraries is an important feature to maintain the homogeneity of the code. This has of course consequences from an industrial point of view: the maintenance of code becomes easier (single code for all platforms). Beyond the use of the Java language adopted in this work, we think that this kind of highly structured portable environment is very promising to develop high level portable computational applications including networking capabilities. The integration of a computational application through the Internet offers new perspectives in the distribution codes not only for educational purposes.

At this stage, we have developed different formulations covering the main domains of computational mechanics: heat conduction, linear elasticity (statics and dynamics), nonlinear elasticity, perfect elasto-plasticity, Stokes flow, Navier-Stokes flow, Darcy flow with free surface tracking. Different kinds of numerical algorithms have been developed: direct and iterative linear system solvers, linear and nonlinear time integrations schemes, Newton-Krylov solvers… Various finite element schemes have been used, including parallel schemes based on the Schwarz domain decomposition method and low order stabilized finite elements. The wide range of applications treated within the proposed approach shows its flexibility. The next challenging step of the developments consists in first treating coupled multi-physics problems and second extending the approach to parallel distributed computing.

References

[1] A. K. Noor, Computational structures technology: leap frogging into the twenty-first century, Computers & Structures, 73, 1-31, 1999

[2] Th. Zimmermann, Y. Dubois-Pèlerin and P. Bomme, Object-oriented finite element programming : I. Governing principles, Comput. Methods Appl. Mech. Engrg., 98, 291-303, 1992.
[3] D. Eyheramendy, An object-oriented hybrid Symbolic/Numerical Approach for the Development of Finite Element Codes, Finite Element Analysis and Design, Vol. 36, 315-334,2000.
[4] 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, 2729-2751, 2001.
[5] 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, 31 (2000) 743-751.
[6] 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, 61-77 (2004).
[7] 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).
[8] D. Eyheramendy, Object-oriented parallel CFD with JAVA, 15th International Conference on Parallel Computational Fluid Dynamics, Eds. Chetverushkin, Ecer, Satofuka, Périaux, Fox, Ed. Elsevier, 409-416, 2003.
[9] J.M. Bull, L. A. Schmith, L. Pottage and R. Freeman, Benchmarking Java against C and Fortran for Scientific Applications, Joint ACM JavaGrande – ISCOPE 2001 Conference, Stanford Universtity, June 2-4, 2001.
[10] G.P. Nikishkov, Y.G Nikishkov and V.V Savchenko, Comparison of C and Java performance in finite element computations, Computer & Structures, 81, 2401-2408, 2003.
[11] J. Häuser, T. Ludewig, R.D. Williams, R. Winkelmann, T. Gollnick, S. Brunett and J. Muylaert, A test suite for high-performance parallel Java, Advances in Engineering Software, 31, 687-696, 2000.
[12] C.J. Riley, S. Chatterjee and R. Biswas, High-performance Java codes for computational fluid dynamics, Concurrency and Computation: Practice and Experience 15, 395-415, 2003.
[13] Y. Dubois-Pèlerin, Th. Zimmermann and P. Bomme, Object-oriented finite element programming : II. A prototype program in Smalltalk, Comput. Methods Appl. Mech. Engrg., 98, 361-397, 1992.
[14] Y. Dubois-Pelerin and Th. Zimmermann, Object-oriented finite element programming: III – An efficient implementation in C++, Computer Methods Appl. Mech. Eng., 108(2), 165-183, 1993.
[15] D. Eyheramendy, Object-Oriented Finite Elements programming in JAVA: I - Basic Principles, Preprint CDCSP 04-01 In preparation, CDCSP Lyon 1 University (2005).
[16] Flanagan, Java in a Nutshell, Fourth edition, Ed. O'reilly, 2002.

[17] S.J. Lacy and J.H. Prevost, Flow through porous media: A procedure for locating the free surface, Int. J. Num. An. Meth. Geom. 11, 585-601, 1987.
[18] D. Eyheramendy and D. Guibert, A Java Approach for Finite Elements Computational Mechanics, ECCOMAS 2004, Jyvaskyla, Finland (2004).
[19] A. Masud and T.J.R. Hughes, A stabilized mixed finite element method for Darcy flow, Computer Methods in Applied Mechanics and Engineering, 191, 4341-4370, 2002
[20] J.C. Simo and T.J.R. Hughes, Computational Inelasticity, Springer, 2000.
[21] Lemaitre and J.L. Chaboche, Mécanique des matériaux solides, Dunod, 1996.
[22] S. Commend and Th. Zimmerman, Object-Oriented Nonlinear Finite Element Programming : a Primer, Adv. In Soft. Engnrg, 32 (8), 611-628, 2001.
[23] D. Eyheramendy, Object-Oriented Finite Elements programming in JAVA: III - Applications, Preprint CDCSP 05-03 In preparation, CDCSP Lyon 1 University (2005).