A JAVA APPROACH FOR F.E. COMPUTATIONAL MECHANICS

Dominique Eyheramendy* and David Guibert*

* MCS/CDCSP, ISTIL Lyon 1 University
15, Blvd. Latarget, 69622 Villeurbanne, France

e-mails: eyheramendy@mcs.univ-lyon1.fr, dguibert@edesp.univ-lyon1.fr

Key words: Object-oriented programming, Java, finite elements, free surface seepage.

Abstract. The object-oriented (O.O.) paradigm is today widely spread out into the finite elements (F.E.) community in computational mechanics. The C++ is the language the most commonly adopted in the developments as support of the O.O. paradigm. Nowadays, developments in computational mechanics involve the use of additional technologies to build global computational systems: graphical components to visualize results, parallelism concepts for large computations... An emerging technology seems to offer the right framework to bypass these cumbersome features: JAVA. The JAVA environment offers the developers to design large scale applications within a single O.O. concept. The environment embeds the basic components needed for modern computational tools through a simple and strong O.O paradigm: homogeneous development of F.E. kernels, development of portable GUIs, developments of distributed applications... This paper is first, an attempt to address some possible directions for the future generation of F.E. codes, and second, to address basic issues of F.E. model integration in JAVA. The problem of numerical efficiency is addressed in complement and an example of classical computational mechanics problems illustrate the high potential of the approach: a free surface unconfined seepage problem.
1 INTRODUCTION

The object-oriented paradigm has been widely adopted in the F.E. computational mechanics community in all fields of computational mechanics. The main reason for this strong interest for O.O 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 O.O oriented paradigm and to be efficient enough to deal with large scale computations. Many computational fields in mechanics have been investigating this approach: e.g. in constitutive law modeling [1,2], in finite deformation plasticity [3], in parallel finite element applications [4-6], in rapid explicit dynamics [7,8], in fracture mechanics [9]. These examples and references therein show that every researcher or 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. But due to its complexity and to its strong dependence on the platform, code developments and maintenance in C++ may lose interest for complex applications, including for example distributed computations. Moreover, the developers need to bring under control multiple computer science technologies (graphical libraries -e.g. OpenGL-, communications libraries -e.g. MPI-...). The approach we propose in this paper is based on a pure Java paradigm. This permits us to overcome the difficulties encountered using a classical C++ framework. The Java language is gaining more and more attention into the F.E community. In [10], the problem of the use of Java for numerical computation in the industrial real life problems is raised up, and no definitive response is brought probably because of lack of experiments in the domain. One aim of the present work is to give an example of large scale development in F.E. for engineering applications. This work relies on previous experiences in O.O. development of F.E. codes in mechanics (see [11,12]). In a first part, a special attention is paid on a description of the Java environment, and a tentative comparison with C++ is done. The second part focuses on the practical problem of F.E O.O implementation in Java. The illustration is given in the third part on a seepage free surface problem.

2 OBJECT-ORIENTED TECHNIQUES FOR FINITE ELEMENTS PROGRAMMING

2.1 Java: an alternative to C++

When developing a new application in computational mechanics, the choice of the programming language remains today strategic. Roughly speaking, the engineer or the researcher has to take into account several parameters such as efficiency, need of high level concepts for fast prototyping, scale of computations, need of multi-processing computation, need of high level graphic user interface,… This choice may become somehow tremendous when multiple constraints have to be considered. The consequence for large scale
computational applications is that multiple programming environment, programming languages and programming paradigms have to be mixed in a single environment. E.g., a single application may:
- be programmed in C++ with an extended object-oriented paradigm,
- include C function and Fortran routines,
- have a graphics user interface including X or OpenGL primitives,
- use a communication library such as MPI or MPICH…

As matter of fact, portability of such an application may become tremendous, and many applications do not offer a distribution overall systems. From an economical point of view, the development of an application requires multiple competences. This kind of strategy has the main advantage to use the most appropriate software solution from an efficiency point of view, but portability may not be ensured. Moreover, modern software development requires today networking capabilities.

Java offers a good alternative to the use of multiple software systems for the development of computational mechanics application. Java is more than a programming language. It offers a real programming platform—so-called Java Platform—, i.e. a large number of predefined classes covering all the aspects of modern application construction. A thorough description of the Java platform goes beyond the purpose of this paper and may be found in [18]. We summarize here its main aspects: graphics user interface, networking, secure application sharing, object persistency, reflection, exceptions handling, multi-threading… Note that in Java, the network is considered as a computer. It means that running a computational mechanics application is conducted on a multi-scale computer system. As matter of fact, it may be interesting to take advantage of it to solve multi-scale numerical problems. To complete the description, an application developed and compiled on a computer system based on a specific Java platform may be run every system, without compilation. This property may be full of interest to ensure portability of a computational code on heterogeneous system, including e.g. GUIs and communication schemes. The code is “Write once, run anywhere”. In fact, the code is compiled and byte code is obtained. The latter is then interpreted by the way of the Java Virtual Machine (JVM). Portability is then achieved on all computer systems on which the JVM has been ported. It is worth noting that many parts of the Java platform are implemented using native machine code. This part of interpretation means that codes are less efficient than pure compiled codes. From a computational efficiency, some experiments has been conducted and show that today usually more than 80% of efficiency of similar C code can be achieved (see [14] for basics comparative tests for a CFD application). In [15-17] similar results were obtained but a few restrictions first on the version of JVM on specific platforms. The last technologies of just-in-time compilers used in the JVM, for which a conversion of byte codes to native machine code is done on the fly, become more and more efficient and will probably soon achieve the performance of classical C or C++ codes.

2.2 Java: a simpler object-oriented programming language

In this paper, we do not want to give a thorough description of the Java language that may be found in the forthcoming paper [13]. We expect the reader to be aware to classical O.O
programming. However, it is important to draw the main tracks for developing in Java in the particular case of computational mechanics. First, the Java basic syntax is similar to the C one. Compare to C++, syntaxes are somehow simplified. For example, the ambiguous use of keywords `goto` and `const` are forbidden. One major difference of the basic syntax comes from arrays manipulation. In Java, an array is an object. It means for example that an array handles its own size, although the syntax is similar to a C or C++ one. A few statements are also added in Java. Let us give two mechanism really useful in computational mechanics:

- as Java is a multi-threaded language; the statement `synchronized` permits the programmer to prevent different threads from modifying an object simultaneously,
- in Java, a mechanism allows the programmer to prevent the from stopping after an error or an exceptional condition; the statements `throw` and `try/catch/finally` allows the global management of the exceptions mechanism.

But, the major difference with the C++ language concerns the O.O organization of codes. Roughly speaking, all the features that make the C++ language difficult, disappear in Java (e.g. the template mechanism). Java support a pure O.O paradigm. It supports only a simple inheritance mechanism (see [18,19] for O.O concepts). The hierarchical organization of the application is then simpler to manage and no ambiguous pattern can happen. The mechanism of interface can supply the lack of multiple inheritance. An interface is simply a pure abstract class that can only mark an other class.

Moreover, the Java language offers richer mechanisms for classes organization than C++:

- different level of inner classes, classes nested in other classes that give various level of visibility and usefulness of classes in the code (4 kinds of inner classes: static member classes, member classes, local classes, anonymous classes –see [18] for detailed descriptions and [13,14] for application in F.E codes),
- classes are organized into packages; a package is a named collection of classes that have a hierarchical organization; moreover, it is a new level of data protection.

To conclude this section, the programmer has a powerful tool to organize classes in order to build its own computational application.

2.3 Object-oriented programming of F.E for computational mechanics

O.O programming has been applied to F.E programming in computational mechanics for 15 years. First in [20,21], some basics methodologies for F.E implementation were proposed: basic linear algebra objects and first reference to basic structural objects such as node and degree of freedom. Modularity and reusability of O.O F.E codes were first discussed in [22,33]. Among the first description of a global O.O framework for structural mechanics, one can find [23,24,29]. This programming paradigm is today widely used in the F.E computational community, including industrial application. More recently, we have proved that this approach may extended to high level mathematical concepts for F.E formulations developments and automatic generation of code [11,12]. In this work, the problem of the programming language was risen up, and the final choice has been to develop complex high level mathematical concept in Smalltalk in order to fasten developments, and to chose a C++ classical F.E code as target to automatic developments in order to get a suitable numerical
efficiency. Ensuing this work, we affirm that the choice of the language to support O.O is very crucial to design large and robust and portable applications. The challenging problems in mechanics will tend in the future to be multi-physics problems more or less coupled, multi-scale and from a practical point of view have to be run on heterogeneous distributed computer systems. Information technology will probably have a strong impact in the design of computational applications in mechanics (see [24] for a discussion about upcoming technologies in computational structures). We advocate that Java is a suitable framework to handle large, complex and numerically efficient computational application. The panel session [10] focuses on the use of Java for intensive numerical computation in Java without definitive response probably because of lack of experiments. Until now, a very few computational F.E for computational mechanics codes in Java exist except and not much attention seems to be paid by the F.E computational community (see the web site www.javagrande.org for a more computational science view). In general, Java is often considered as a language adapted for GUIs or communication schemes. We think that Java may also be a real environment for numerical computations. Among the most advanced work relating this kind approach, one may find many applications in CFD. In [25,26], a specialized CFD code for the simulation of heat trasfert and multiphase fluid flow problems is presented. Nonlinear Jacobian free Newton-Krylov solution schemes for finite volume solvers is presented. The application, including share and distributed memory, exhibits good numerical performances. The aim of the project is here to get a flexible software environment for nonlinear physics problems on unstructured grids, and the Java language is proved to be the right framework for it. In [27], special attention is paid on efficient implementations using the Java language. From the point of view of the authors, it seems necessary today to pay attention to the programming syntax. Still application in CFD are concerned. In [28], the capabilities of Java application through the Internet is put in prominent position. An applet for high-order asymptotic analysis of crack-tip fields has been developed. This is a good example of the use of application portability across the web.

The global aim of our work is to develop a flexible portable and parallel framework for complex nonlinear multi-physics problems, based on finite elements solution schemes. User-friendly GUIs for pre- and post-processing should help a basic user to define easily its own problems. The environment will take advantage of the Java platform of object persistency to keep a computational domain in a given state. The first step of this work is very promising. In the next part, we show a typical O.O implementation in Java for physical formulation illustrated on a free surface seepage problem.

3 APPLICATION OF OBJECT-ORIENTED FINITE ELEMENTS PROGRAMMING IN JAVA TO A SEEPAGE PROBLEM

In the previous section, particular idioms of O.O programming in Java were described. In this section, 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 may be interesting to share partially global and local aspects of a numerical scheme. The approach is illustrated on a simple interface tracking
scheme for underground water flow that may be found in [30]. The management of the finite element formulation is described on this example.

3.1 Free surface seepage problem

A procedure to locate precisely the free surface of unconfined seepage flow through porous media is described in [30]. Let us briefly recall the problem. We consider the flow of one 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 $\Omega_w$ represents the flow region (the saturated part of the geometric domain $\Omega$ occupied by the earth structure) and $\Omega_d$ the dry part.

![Figure 1: Definition of the free seepage problem.](image)

The piezometric head $\Phi$ is defined with respect to the pressure $p$ as:

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

where $\rho$ is the density of the fluid and $\gamma$ the specific weight of the fluid.

The steady state problem may be modeled as follows:

$$p \geq 0 \text{ in } \Omega_w$$  \hspace{1cm} (2a)

$$p = 0 \text{ in } \Omega_d$$  \hspace{1cm} (2b)

$$\nabla \cdot k \cdot \nabla \phi = 0 \text{ in } \Omega$$  \hspace{1cm} (2c)

where $k$ is the Darcy permeability tensor.

The boundary conditions play a crucial role for this kind of problem and are expressed:

$$p = \gamma (H_1 - y) \text{ on } AF$$  \hspace{1cm} (3a)
where \( n \) is the outward unit vector defined on the domain \( \Omega_w \).

### 3.2 Weak formulation and mathematical procedure

The weak form of the problem can be written as:

\[
\begin{aligned}
\text{Find } p \in P = H^1(\Omega_w) \text{ such that } \forall q \in H^1_0(\Omega_w) \\
\int_{\Omega_w} k \nabla p \nabla q \, d\Omega = -\int_{\Omega_w} \frac{1}{\gamma} k \nabla y \nabla q \, d\Omega
\end{aligned}
\]  

(4)

The fundamental difficulty is that the location of the free surface is unknown a priori and enforcement of the boundary conditions on the free surface may be tremendous. A simple weak formulation was proposed in [31]. 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 \( \Omega_d \) -the dry soil domain-. Existence and uniqueness of the solution for the extended problem is showed by introducing the following penalized problem:

\[
\begin{aligned}
\text{Find } p_\varepsilon \in P = H^1(\Omega_w) \text{ such that } \forall q \in H^1_0(\Omega_w) \\
\int_{\Omega_w} \frac{1}{\gamma} k \nabla p_\varepsilon \nabla q \, d\Omega = -\int_{\Omega_w} k H_\varepsilon(p) \nabla y \nabla q \, d\Omega
\end{aligned}
\]  

(5)

where \( H_\varepsilon(p) \) is an extension of the Heaviside function defined by:

\[
H_\varepsilon(p) = \begin{cases} 
1 & \text{if } p \geq \varepsilon \\
\frac{p}{\varepsilon} & \text{if } 0 \leq p \leq \varepsilon \\
0 & \text{otherwise}
\end{cases}
\]

(6)

Since the location of the free surface is unknown a priori, a Newton iterative scheme is required to solve the problem. The same stopping criteria as the one found in [30] are used to solve the penalized problem. A local computation (at the elemental level) is done to determine of the penalty parameter \( \varepsilon \) and a cut-off pressure \( p_a \) which enforce the pressure to a small negative in the dry soil. The discretized form of the variational problem (5) may be written as follows using a simple linearization technique (see [32]):
The elemental contributions may be written as follows as printed in table 1 (A denotes the assembling operator and the $N_a$ are the shape functions expressed for a given node $a$).

\[
\begin{align*}
K, \partial p^n &= f - Kp^n \\
p^{n+1} &= p^n + \partial p^n
\end{align*}
\]

(7)

Table 1: Elemental matrices for the penalized free surface seepage problem

3.3 Implementation in the Java code

The main object in our FEM code 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 builds only one scalar field, the pressure. And on the mesh, F.E. elements $\Omega$ are built, they were able to give the elementary constitutive matrices such as $k_e^e, k_e^c, f_c^c$.

The implementation of the formulation is made through a class called: **PressureDarcyPenalizedProblem**. The class is posted in table 2. It belongs to a package called **femjava.fem.formulation.basics**. The class subclasses **Formulation** in which all the basic behavior of the F.E formulation is taken into account, e.g. the global way of building fields overall the computational domain. The class embed an inner class called Darcy; this a static member class. The two important methods of 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
- **getElement()** which permits us to instanciate 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** is defined as an inner static class (a kind of class of class) and exhibits as local behavior all what is needed to compute the finite element matrices defined table 1. This is a subclass of **Element** which processes all the behavior mandatory to deal with elemental data: nodes local to an element, finite element interpolations... The
definition of one class and its inner class permits us to completely define a new finite element formulation. Global solution algorithms are programmed in a general way. This special feature of the Java programming language can be seen as a generalization of the O. O concept at the level of a class. This leads to a better organization of code.

```java
package femjava.fem.formulation.basics;
// Packages inclusion

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 …
    }
}
```

Table 2: Implementation of an inner class in Java for defining a formulation

### 3.3 Numerical examples

For all the test cases, the permeability $k$ is set to identity tensor and the specific weight of water $\gamma$ is set to 1.0. Bilinear quadrilateral elements have been used. For all the numerical tests, piezometric heads are indicated on figures 2-3-4. We obtained similar results to the one of [30]. The test shown figure 2 is an homogeneous dam. The solution exhibits a surface of seepage. In the second test (figure 3), the dam has a toe drain. In the last test (figure 4), the dam has 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 $\varepsilon$, cut-off pressure definition, penalty parameter to impose the zero-pressure overall the domain. The test shown figure 2 converges in 7 iterations, the two following in 13 iterations. The purpose of this work was for us to rapidly get aware about this formulation and to check its performances. We have noted that the precision of the solution depend drastically on the choice of various parameter involved in the formulation. The way to imposed the boundary conditions for the pressure especially on the free surface seepage zone may lead to instable results. These experiment 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.

![Figure 2: Dam with tailwater.](image1)

![Figure 3: Dam with toe drain.](image2)

![Figure 4: Dam with impermeable sheet on upstream face.](image3)

12 CONCLUSIONS

In this paper, we have presented some basics aspects of an object-oriented paradigm for finite elements using a Java approach. A comparison with the C++ language show that this
new language is much more simpler and offer a wider range of data organization. It allows the programmer to progressively share information with other entities of the code. Package organization of code leads to a natural library type organization of code. By the way of the JVM codes can be run on heterogeneous computer system, and object are easy either to transfer over the network or to save on a hard disk. The distribution of an application through an Internet browser is seamless; this offers a new perspective in the distribution of F.E applications not only for educational purposes. Although a particular attention must be paid to achieve good performances, Java opens new promising tracks to design modern computational mechanics applications. We illustrate a feature of data organization in Java on a simple problem of free surface seepage problem: an inner class implementation. This illustrate the high potential of complex data structuring in Java. At this stage of the development, we have developed different formulations: heat conduction, linear elasticity, nonlinear elasticity, perfect elasto-plasticity, Stokes flow, Navier-Stokes flow, Darcy flow with free surface tracking. The wide range of the applications treated here shows the flexibility of our approach. Multiple resolutions algorithms have been developed: direct and iterative linear system solver, linear and nonlinear time integrations schemes, various Newton-Krylov solvers… Various F.E schemes have been used, including parallel schemes. A portable GUI for generic tools for pre- and post-processing has been developed. The next challenging step of the developments will be to treat coupled multi-physics problems and to extend the parallel computations to distributed computing.

12 ACKNOLEGDEMENTS
This work was partially supported by the Région Rhône-Alpes under: “Thématique prioritaire mathématiques” and “Fond d’Investissement et de Transfert Technologique 2003”.

REFERENCES


