12th International Research/Expert Conference
"Trends in the Development of Machinery and Associated Technology"
TMT 2008, Istanbul, Turkey, 26-30 August, 2008

OBJECT-ORIENTED PROGRAMMING APPROACH IN THE FINITE ELEMENT MODELING

Salko Cosic, University of Tuzla, Bosnia and Herzegovina
Faculty of Mechanical Engineering, Univerzitetska br.4, 75000 Tuzla, BiH
salko.cosic@untz.ba
Mevludin Avdic, University of Tuzla, Bosnia and Herzegovina
mevudin.avdic@untz.ba

ABSTRACT
In this paper we present advantages of object-oriented programming concept with respect to conventional, procedural approach for the case of finite element modeling in nonlinear structural mechanics. We analyze and discus the program structure, main classes and implementation details. The accuracy and efficiency of object-oriented program design is evaluated by comparing with procedural program for the same task.
Keywords: Object-oriented programming, Modeling, Numerical methods, Nonlinear finite element, C++ Classes implementations.

1. INTRODUCTION
Finite element method is become the main numerical technique for the modeling of complex nonlinear problems in the continuum mechanics, particularly in the structural mechanics. Since its beginning, it is implemented in classical procedural program codes. Classical programming approach in finite element programming has a high efficiency. For modern software, efficiency is important but, also, possibilities for program modification or extension is necessary feature. It allows easy modification or extension of program, use the current finite element technology and benefits of new computational techniques. In contrast to OO concept, conventional, procedural concept has a simple up-down structure, fig.1. But, unfortunately, in procedural programming concept, extension and modification of source code is risky, difficult and frustrating. The main problems come from:

- Modification of codes is possible only for programmer specialists, very familiar with data structures and implemented methods. If source code consists of several thousands lines and if it is poorly documented (commented), modification is quite difficult.
- Possibilities to use subroutines (functions) from other programmers and sources are limited be cause of different program structure, declarations of variables and so. It is necessary to be very careful when including external source code in the base application.
- Nonlinear finite element analysis with material nonlinearities requires tracking of history variables (internal variables) for every integration point (GP) in each finite element. As the number of iteration is unknown in advance, declaration, space allocation and manipulation with such (large) data arrays could be troublesome.
- Re-meshing (adaptive meshing) is very difficult to implement due to fixed data structure.

Current development in FE software design is the related to two main observations. The first concerns the growing needs of modeling complex interaction phenomena between different physical fields (e.g. thermo-mechanical coupling), and the second is the evolution of new computing technology which

invites developers to reconsider the new language features or by proposing adapted algorithms to distribute tasks between computers connected in parallel.

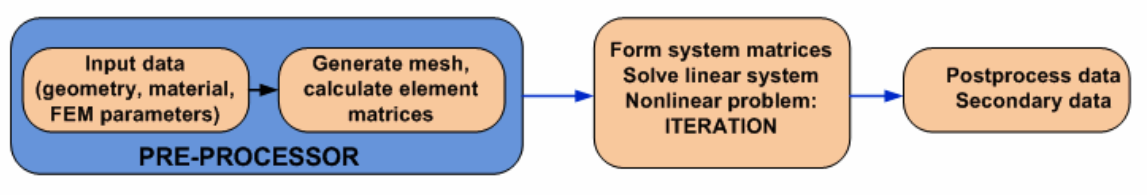

Figure 1. Conventional Finite element programming concept

2. OBJECT ORIENTED PROGRAMMING CONCEPT IN FINITE ELEMENT MODELING

Nowadays, large part of technical software is designed and implemented in Object-Oriented fashion. Especially, graphical and data-bases applications which are standard parts of finite element analysis system are designed in Object-Oriented way. A modern finite element application deals with complex and sophisticated problems with lots of individual sub problems. The basic entities of the finite element method like elements, nodes, materials, boundary conditions, etc. can be modeled by objects, abstract data types with their own behavior, state and identity. The mathematical, physical, numerical, geometrical and programming tasks are implemented in different classes, objects and numerical methods. The derived objects are capable of performing predefined procedures, storing information (in its internal variables), executing its own methods or accessing other objects by sending messages. Variables describe the state of the object, while methods define its behavior. Objects hide their variables from other objects in the same application. For instance, class Element does not have access to its material properties. Material data are stored in class Material. When an element needs material data for the computation of its own stiffness matrix, it has to send a message like "giveMaterial_Data()" to an object of "Material" Class. Method "giveMaterial_Data ()" sets then the proper way to respond to the message, and returns the requested value to the element. Object-Oriented (OO) programming approach allows:

- **Robustness and modularity: encapsulation of data**
- **Inheritance and polymorphism: the hierarchy of classes**

A class is an abstract data type which can be considered as the example of the object. Classes are organized within a hierarchy (super-class-subclass), which allows a subclass to inherit the methods and variables from its super-class. For example, class Trus2D can inherit methods and variables of its super-class, class Element. The inheritance feature allows the programmer to define the common functions and data used by several classes at the highest possible level in the hierarchy which avoids the duplication of code at the lower levels. The descendent classes may add additional attributes and methods and can redefine the methods of an ancestor class. The inheritance is provided to build hierarchical structure of objects and enable code reuse.

Object-oriented programming is primarily a data abstraction technique. The purpose of abstraction and data hiding in programming is to separate behavior from implementation. For abstraction to work, the implementation must be encapsulated so that no other programming module can depend on its implementation details. In OO programming, further flexibility for element formulation can be obtained through the polymorphism feature. Polymorphism expresses the fact that two different classes will react differently to the same message. For instance, the message "giveBMatrix()" will be interpreted differently by an object of the class Quad_EL (defining quadrilateral elements) and an object of the class Truss2D (defining truss elements). For an implementation the following are provided for each class:

- class interface defining the methods that can be invoked on each object of the class
- private data defining the attributes held privately by each object
- method implementations code defining the sequence of operations that objects of the class perform in order to complete the method invoked on the object.

3. OBJECT ORIENTED FINITE ELEMENT PROGRAM STRUCTURE

The process of solving a continuum mechanics problem regularly involves a discretization step followed by a step of computation of system matrices, finding the solution to a discrete problem (solving system of linear equations) and results post processing. For example, the structural mechanics problem in linear statics is expressed by a boundary value problem which is, after discretization, transformed into a system of linear algebraic equations. When dealing with dynamical analysis, we have to solve a system of linear differential equation (eigenvalue problem). In nonlinear statics analysis (material or geometrical nonlinearity) the problem to be solved is a system of nonlinear algebraic equations. Therefore, the program structure of O.O. finite element program is presented on fig.2.

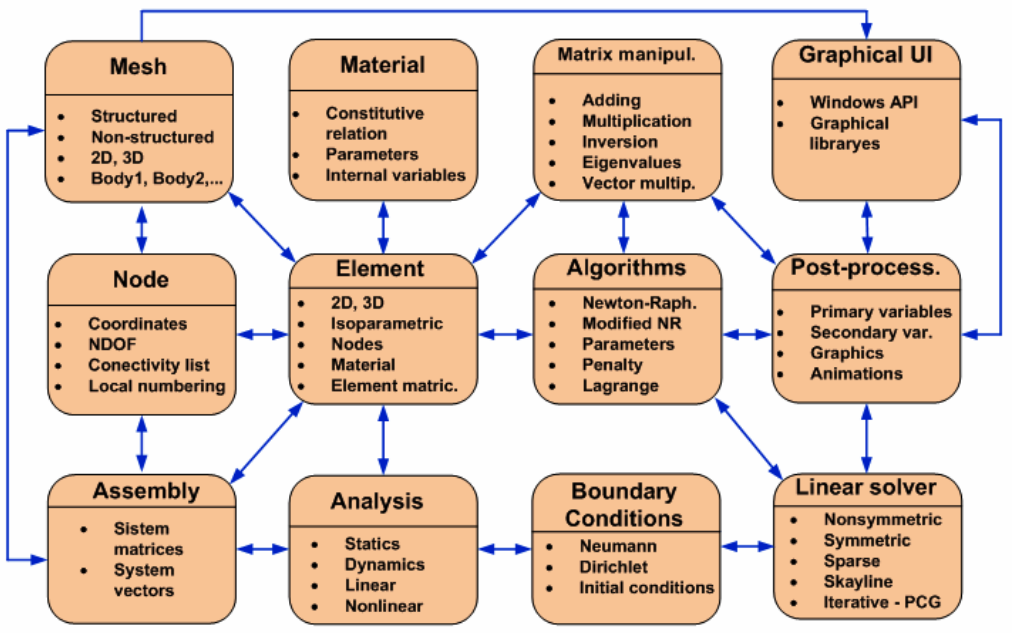

Figure 2. Object oriented finite element programming concept

The main classes are: mesh, element, node, boundary conditions, algorithms, linear solver... etc. Domain Discretization into finite elements method is the first step. This is done with the Mesh class. Finite element mesh is an instance of class Mesh that can be constructed by simply giving a mesh geometry data to Mesh class. An object of "Mesh" class has to communicate with objects of "Node" and "Element" classes as well as GUI and "Assembly" classes. Data necessary to define to the class Element may include such items as localization, connected nodes, number of degrees-of-freedom, etc. Each Element object is associated with a number of Node objects_ as shown in figure 2. Methods are provided to allow other objects in the program to obtain information about the number of Nodes and the Node identifiers. Methods are provided to return the current linearized mass, stifness and damping matrices. Also, in case of nonlinear analysis, methods are provided to return element load vector and its contribution to residual force vector.

Another important class is the „Material" class. Since there can be several different types of material models, this is a superclass and then there are several subclasses (elastic materials, elasto−plastic materials, soil, rocks, visco-elastic materials, rubber...).

A basic step in the implementation of the finite element method consists in building up finite element equations for each element and assembling them into a global linear system of equations. Of course, a nonlinear problem is solved by an iteration algorithm where each iteration consists in solving a linear problem. The same technique is valid for transient problems where iterations are performed in order to advance in time. Here at the element level, finite element equations are to be declared as instances of an "Analysis" class. After implementation of Boundary Conditions (BC), class "Linear solver" solves the resulting algebraic system. Obviously, the type of finite element equation depends on the type of the problem to solve.

4. CONCLUSIONS

In this paper we describe an object-oriented finite element program structure for nonlinear structural and continuum analysis. The advantages of object-oriented approach to solving nonlinear finite element problems have been demonstrated. Object-oriented finite element programs are implemented mainly in C++ and Java programming languages. It has been shown in practice that improper implementation of C++ programs is the consequence of a reduced understanding of some language features. It must be borne in mind that abstraction often comes with a loss of computational performance. Correctly written OO codes in C++ have been shown to have efficiency close to classical Fortran codes. In terms of pure computational efficiency, J.J. Dongarra and R. Pozo obtained with the Lapack++ library (a C++ extension of the famous Fortran Lapack library for numerical linear algebra) exactly the same performance as with Fortran. On the other hand, the C++ implementation provides:

- reusability,
- extensibility,
- scalability,
- portability and
- flexibility.

Also, integration in operation system is easier. It is important to emphasize, the flexibility does not come by sacrifiyng the structure or simplicity of the code. The "code-reuse" and "programming by specification" can be repeatedly applied. Special attention has to be focused on graphical interface design, analysis type and implemented numerical methods. Successful implementation using C++ programming language verifies the designed structure and provides a robust computational tool for finite element modeling.

5. REFERENCES

[1] A. Cardona, I Klapka, P Devloo: „OO programming in FE analisys", Academic Press 2001

[2] Barton J, Nackman L Scientific and Engineering, C++. Addison Wesley, (1994).

[3] Cardona A, Klapka I, Geradin M (1994) Design of a new finite element programming environment. Engineering Computations 11:365-81.

[4] Cary JR, Shasharina SG, Cummings JC, Reynders JVW, Hinker PJ (1997): „Comparison of C++ and Fortran 90 for object-oriented scientific programming", Computer Physic Communications 105:20-36.

[5] Coad P, Yourdon E.: „Object-Oriented Design. Yourdon Press Computing Series" Devloo PRB 1994.

[6] Metcalf M, Reid J.: Fortran 90 Explained. Oxford: Oxford Science, 1995.

[7] Miller GR. : „An object-oriented approach to structural analysis and design", Computers and Structures 40:75 - 82.

[8] Pantale O., Caperaa S., Rakotomalala R.: "Development of an object-oriented finite element program: application to metal-forming and impact simulations", Journal of Computational and Applied Mathematics, Vol. 168, No. 1-2, p. 341-351

[9] Ph. Menétrey, Th. Zimmermann: Object-oriented nonlinear finite element analysis: application to J2 plasticity, Computers & Structures, 49, pp. 767-777, 1993

[10] Y. Dubois-Pélerin, Th. Zimmermann, P. Bomme - Object-oriented finite element programming: II A prototype program in Smalltalk, Comp. Meth. Appl. Mech, Eng. (98), 1992

[11] Y. Dubois-Pélerin, Th. Zimmermann - Object-oriented finite element programming: Theory and C++ implementation for FEM_Object C++ 001, Elmepress International, 1993

[12] Y. Dubois-Pélerin, P. Pegon - Object-oriented programming in nonlinear finite element analysis, Computers & Structures, 67, pp. 225-241, 1998

[13] T.J.R. Hughes - The finite element method, Prentice-Hall, 1987

[14] T.J.R. Hughes, T. Belytschko - Nonlinear finite element analysis, Paris Short-Course Notes, 1997

[15] Th. Zimmermann, Y. Dubois-Pélerin, P. Bomme - Object-oriented finite element programming: I Governing principles, Comp. Meth. Appl. Mech, Eng. (98), 1992

[16] Zeglinski GW, Han RPS (1994) Object-oriented matrix classes for use in a finite element code using C++. International Journal for Numerical Methods in Engineering 37:3921-3937.