Mathematical Modelling and Numerical Analysis
Modélisation Mathématique et Analyse Numérique

ESAIM: M2AN
M2AN, Vol. 36, N° 5, 2002, pp. 773–781
DOI: 10.1051/m2an:2002044

FER/SUBDOMAIN: AN INTEGRATED ENVIRONMENT FOR FINITE ELEMENT ANALYSIS USING OBJECT-ORIENTED APPROACH

Zhi-Qiang Feng[1] and Jean-Michel Cros[1]

Abstract. Development of user-friendly and flexible scientific programs is a key to their usage, extension and maintenance. This paper presents an OOP (Object-Oriented Programming) approach for design of finite element analysis programs. General organization of the developed software system, called FER/SubDomain, is given which includes the solver and the pre/post processors with a friendly GUI (Graphical User Interfaces). A case study with graphical representations illustrates some functionalities of the program.

Mathematics Subject Classification. 68N99, 65M55, 68U05, 74S05.

Received: October 8, 2001. Revised: May 13, 2002.

Introduction

The finite element method is a powerful technique for the solution of complex engineering problems. One of the significant requirements in the design of a finite element structural analysis program is the ability to store, retrieve, and process data that maybe complex and varied. To the users of such program, it is important not only to have a powerful finite element solver, but also to work in a convivial graphical interface environment. On the other hand, as the problems to solve have grown in size and complexity, the codes have also grown with complex mathematical procedures and data control. This places a high demand for maintenance, new developments and re-use on the programming strategy and language chosen.

OOP (Object-Oriented Programming) is a topic well-known to computer scientists, but somewhat neglected in the computational engineering community. One reason for this is the limited exposure of engineers to computer science concepts. Another reason (historical reason) is that most finite element analysis programs have been and are being written in a procedural programming language such as FORTRAN. Because of its design, FORTRAN does not encourage the use of data structures other than the array. The object-oriented programming techniques are not supported by the language itself. Consequently, the analysis programs are not easily modified for implementing new ideas and new algorithms. However, since several years, this problem has come to the attention of the engineering profession, and much progress has been made to improve the reliability of methods for finite element analysis and to make it easier for usage, extension and maintenance of analysis programs.

Keywords and phrases. Object-oriented programming, domain decomposition, graphical interface.

[1] Centre d'Études de Mécanique d'Ile de France, Laboratoire de Mécanique et Énergétique d'Évry (CEMIF-LMEE), Université d'Évry-Val d'Essonne, 40 rue du Pelvoux, 91020 Évry Cedex, France. e-mail: feng@iup.univ-evry.fr

In the eighties, several researchers began work on data management in structural analysis software [5,16,17, 23,26,32]. In 1986, Touzot [30] introduced an interactive conception system SIC. One year after, De Saxcé [6] presented the project CHARLY. Verpeaux *et al.* [31] presented the CASTEM finite element program. These three programs aimed at providing a veritable language devoted to finite element modeling, based on the object database concept. One of the first detailed applications of the object-oriented paradigm to finite element analysis was published in 1990 by Forde *et al.* [15]. The authors abstracted out the essential components of the finite element method (elements, nodes, materials, boundary conditions, and loads) into a class structure used by most subsequent authors. Also presented was a hierarchy of numerical objects to aid in the analysis. Other authors [14,21,25,27] increased the general awareness of the advantages of object-oriented finite element analysis over traditional FORTRAN based approaches. Some researchers have concentrated on the development of numerical objects. Scholz [28] gives many detailed programming examples for full vector and matrix classes. Zeglinski *et al.* [34] provide a more complete linear algebra library including full, sparse, banded, and triangular matrix types. Also included is a good description of the semantics of operators in C++. Lu *et al.* [20] present a C++ numerical class library with additional matrix types such as a profile matrix. They report efficiency comparable to a C implementation. Zimmermann *et al.* [7,8,35] have developed a software architecture in C++ and in SmallTalk for linear dynamic finite element analysis, with extensions to account for material nonlinearity [22]. A freeware, named FreeFem+, was proposed by Pironneau *et al.*[1] for solving Partial Differential Equations in 2 dimensions. To our knowledge, the first large scale finite element analysis program, named ZéBuLoN, entirely rewritten in C++, is presented in 1993 by Aazizou *et al.* [1,12].

All the developments mentioned above are concentrated on numerical computation. Recently, several finite element analysis programs including GUI environment such as FER/View, FER/Mech, FER/Contact, etc. have been developed by Feng *et al.* and are reported in a WEB site[2]. This paper presents the development of the program FER/SubDomain. It is composed of several functional modules: the finite element solver associated with the domain decomposition method, the pre and post processors and domain partitioner METIS developed by Karypis *et al.*[3]. Section 2 presents the object-oriented approach in developing finite element program. Section 3 describes the general organization of FER/SubDomain and its main features. Section 4 gives a case study to illustrate some functionalities of FER/SubDomain, described in Section 3. The primary goal of this paper is to illustrate the practical application of the object-oriented approach to finite element analysis.

1. Object-oriented programming in C++

This section introduces some concepts and terminology of object-oriented programming. The basic concept of object-oriented programming is the encapsulation of data structure and a set of functions (procedures) manipulating the data in prepackaged software components called objects. By using an object-oriented language such as C++, a natural way of manipulating finite element objects such as node, element, boundary conditions, matrix, vector, can be adopted. The notion of "object" has been widely employed in many computer science fields. As compared with the traditional function-oriented programming technique, object-oriented programming is more structured and modular, yielding programs that are easily maintained, resilient, and powerful because of its basic features: data abstraction, encapsulation and data-hiding, modularity, classes, hierarchy and inheritance, polymorphism and dynamic binding etc. However, this notion is not largely used in the field of numerical simulation. We know that most programs in scientific computing are written in FORTRAN, in which it is difficult to write structural and object-oriented programs even a concerted effort has been made in this field. Of the many possible programming languages, C++ is being increasingly used in engineering applications because it was designed to support data abstraction structure and object-oriented programming. In addition, C++ is the extension of the popular language C. So it becomes the first choice for scientists and engineers to develop object-oriented programs for analysis of engineering problems.

[1] http://www.ann.jussieu.fr/∼pironneau/freefem.htm

[2] http://gmfe16.cemif.univ-evry.fr:8080/∼feng/

[3] http://www-users.cs.umn.edu/∼karypis/metis

```
class CPartitionInterface {
private:
   int m_numero;
   int m_nodenum;
   VECTINT m_nodeid;
public:
   void setPartIntfNode(int ipart,int num,VECTINT& nodeid);
   int getOpposite PartNumber( ){ return m_numero; }
   int getIntfNodeNum( ){ return m_nodenum; }
   int getNodeId(int i){ return m_nodeid[i]; }
   CPartitionInterface();
   virtual ~CPartitionInterface();
};
```

FIGURE 1. PartitionInterface class definition.

```
class CFERSubDomainApp : public CWinApp {
public:
   CFERSubDomainApp();
   //ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CFERSubDomainApp)
     virtual BOOL InitInstance();
     virtual BOOL PreTranslateMessage(MSG pMsg);
   //}}AFX_VIRTUAL
   // Implementation
   //{{AFX_MSG(CFERSubDomainApp)
     afx_msg void OnAppAbout();
     afx_msg void OnOptionInfo();
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
};
```

FIGURE 2. Derived classes.

The benefit of object-oriented approach in C++ is mainly due to the definition of classes. A class is defined by the groups of objects that have the same kind of data and procedures. Objects are called instances of a class in the same sense as standard FORTRAN variables are instances of a given data type. The class concept can be reviewed as an extension of the record concept in Pascal or struct concept in C to provide a set of attached procedures acting on the data. Unlike conventional programming techniques, which require the developer to represent data and procedures separately, an object in C++ is an user-defined and self-contained entity composed of data (private or public) and procedures. This allows developers to design objects which know how to behave. Figure 1 shows an example of class which defines the interface of subdomain.

In this class, m_numero and m_nodenum are private data of type integer, and m_nodeid is an instance of the class VECTINT that is another user-defined class defining an integer vector. SetPartIntfNode, getOppositePartNumber, etc. are member functions (procedures) of object CPartitionInterface. CPartitionInterface() and ~CPartitionInterface() are respectively the default constructor and destructor. A high-level object can be created by assembling a group of objects. This new object has the collective functionality of its sub-objects. This concept of object abstraction can be extended to the complete software applications. A program assembled from

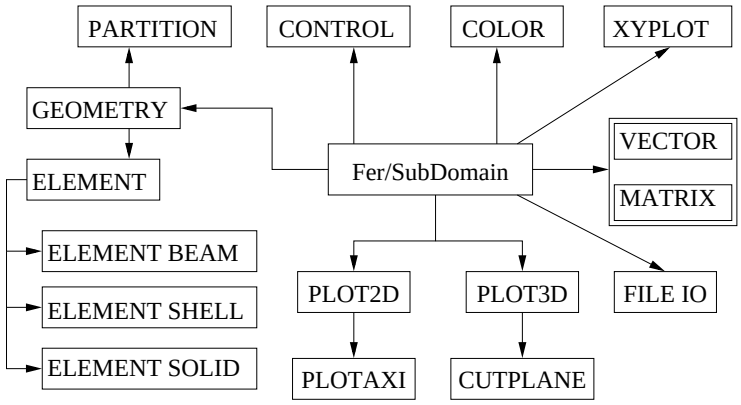

FIGURE 3. Principal class diagram of FER/SubDomain.

objects can itself be an object and thus be included in another program. This method has been applied when adding the domain partitioner METIS into FER/SubDomain.

One of the fundamental techniques in object-oriented programming is the use of inheritance. Inheritance is a way of creating new classes called derived classes, which extend the facilities of existing classes by including new data and function members, as well as changing existing functions. For instance, the development of FER/SubDomain requires the creation of class CFERSubDomainApp which is the derived class of MFC class CWinApp for Windows applications (Fig. 2).

It is noted that objects communicate with each other by sending messages. When an object receives a message, it interprets that message, and executes one of its procedures. That procedure operates on the private data of the object. So the internal details of how it functions are hidden from the program that uses the object. This also means that the internal function of an object could be modified without having to change the rest of the program. Thus, the program becomes modular and easy to read. Without entering into the details, Figure 3 shows the principal database mapping of FER/SubDomain.

CONTROL stores some control flags about light, plot symmetry, animation, cutting plane, dynamic rotation or move, etc. COLOR stores the default color date and procedures for contour, iso-line and iso-surface plot. ELEM_BEAM, ELEM_SHELL and ELEM_SOLID are inherited from the base class ELEMENT.

It is worthy noting that many components of FER/SubDomain were taken from the postprocessor FER/View developed previously [13]. Object-oriented programming makes this possible and allows rapid development of new software. For instance, the development of FER/SubDomain has only taken about one month for two developers. Reusability is becoming a key issue in software development.

2. General organization of FER/SubDomain

FER/SubDomain is an integrated environment composed of several functional modules. Figure 4 shows the flow diagram of the software.

The mesh generator is a third-part software which can be some CAD or finite element programs. The preprocessor imports the mesh (nodes and element topologies). The boundary conditions and the loads can be applied to the model interactively using dialog boxes and selection facilities. METIS is called to perform domain partitioning. After preprocessing, an input file is created, which is used by the solver FER/Parallel. The results from FER/Parallel are written in an output file and displayed by the postprocessor FER/View.

FER/Parallel is a finite element structural analysis program [3, 4] designed to be scalable on parallel computers. Currently the code offers static analysis, direct implicit transient analysis and eigenvalue analysis. These analyses lead to the numerical solution of large scale, sparse and often ill-conditioned linear systems. For solving these linear systems, FER/Parallel uses non-overlapping multilevel domain decomposition methods

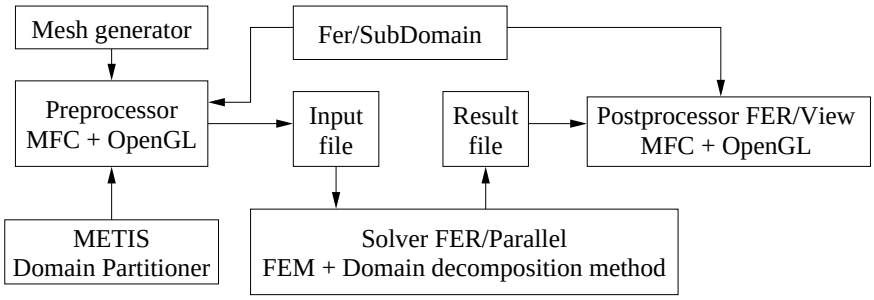

FIGURE 4. Flow diagram of FER/SubDomain.

such as FETI (Finite Element Tearing and Interconnect) methods [10, 11] and primal Schur complement methods [18, 19]. The solution methods are selected based on the criteria of robustness, accuracy, scalability and efficiency. Domain decomposition methods have received much attention in the last few years because they can mix direct solvers used at the subdomain level and iterative solvers used at the interface level. Moreover, these techniques are well-suited to modern parallel computers, because of the built-in parallelism of the algorithm and the good localization of the associated data. The SPMD (Single Program-Multiple Data) model is used for such computations. The parallelism is achieved by partitioning the mesh and by allocating the disjoint subproblems or subdomains to the parallel processors. During each iteration the processors perform (i) an exchange of local data (interface unknowns) with the processors that handle geometrically adjacent subdomains in order to enforce continuity requirements of the solution (local synchronization/communication); (ii) an execution of matrix-vector operations (local computation); and (iii) an evaluation of stopping criteria and acceleration of the convergence (global synchronization). The high performance of these computations on distributed memory MIMD (Mutiple Instructions-Mutiple Data) machines, depends not only on the minimization of the local and global communication time, but also on the synchronization delays. Global communication/synchronization depends on the efficient hardware/software implementation of broadcast operations of parallel machines. The local communication time depends both on data partitioning characteristics such as, interface length, degree of connectivity of the subdomains, and machine characteristics such as the interconnection network and routing. There are a large number of decomposition schemes that have been published in recent years [9, 24, 29]. Generally, the schemes fall into two distinct categories: logical methods that use only connectivity information and physical methods that use only the physical position of nodes. For the moment, in FER/SubDomain we have implemented the first technique by using METIS mesh partitioner. FER/SubDomain saves a specified mesh partition into an ASCII file including the parallel data structures needed for local computations and for message passing. Moreover, in order to reduce the memory required by the factorization of matrices in each subdomain, FER/SubDomain includes a renumbering option to minimize the bandwidth size. FER/Parallel is programmed in C++ for forming stiffness and mass matrix and in FORTRAN for solving linear systems. BLAS and LAPACK libraries are used to improve the efficiency of solution. The parallel implementation of the code has been developed with MPI (Message Passing Interface) to perform the communications.

MFC (Microsoft Foundation Class) [2] has been proposed by Microsoft for easy development of Windows applications. In this project, MFC is largely used to design the user-interface objects such as Dialog boxes, Menu, Icons, Tool Bar, String Table, etc. OpenGL [33] is a relatively new industry standard that in only a few years has gained an enormous following. It is now a standard graphics library integrated in Windows or UNIX systems. OpenGL is a procedural rather than a descriptive graphics language. Instead of describing the scene and how it should appear, the programmer actually describes the steps necessary to achieve a certain appearance or effect. These steps involve calls to a highly portable API (Application Programming Interface) that includes approximately 120 commands and functions. These are used to draw graphics primitives such as points, lines, and polygons in three dimensions. In addition, OpenGL supports lighting and shading, texture mapping, animation, and other special effects. A lot of these capabilities have been implemented in FER/SubDomain.

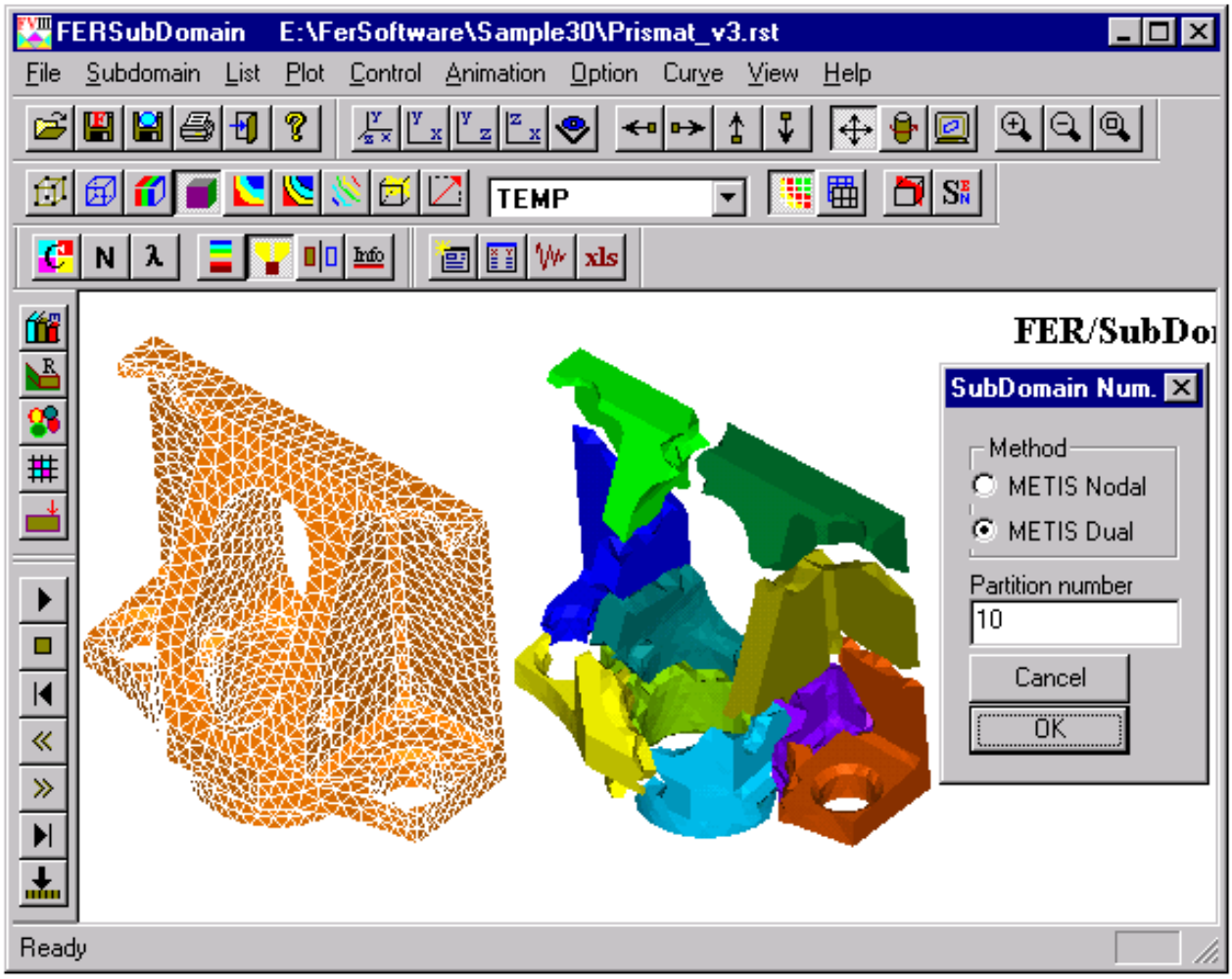

FIGURE 5. Friendly user-interface of FER/SubDomain.

The result is satisfactory. The user-interface of the program is shown in Figure 5, which shows equally an unstructured 3D solid mesh partitioned into 10 subdomains. FER/SubDomain has many functionalities, some main features being summarized as follows:

Preprocessor

- Import nodes and elements from other CAD software (Pro/Engineer, I-DEAS, etc.);
- Display node, element and geometry with or without numbering;
- Input material properties, real constants, solution control parameters, etc. with dialog boxes;
- Select nodes, surface elements and volume elements with dialog boxes and mouse operations;
- Define element attributes and apply boundary conditions to the selected groups;
- List nodes, elements, materials;
- Decompose the mesh into subdomains;
- Create an input file for the solver.

Solver

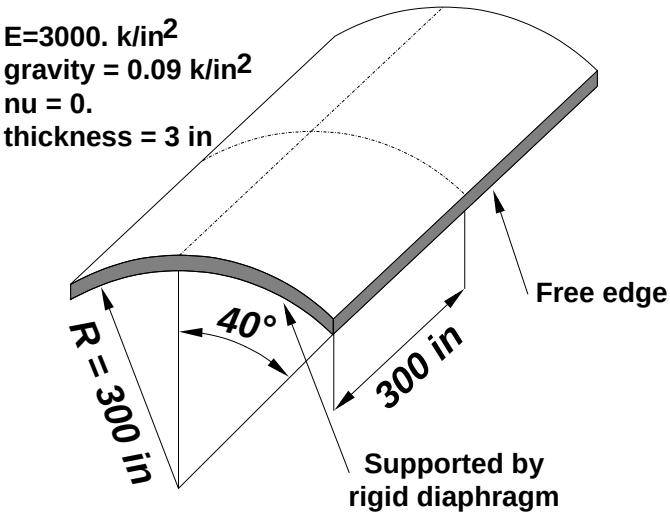

FIGURE 6. Cylindrical roof.

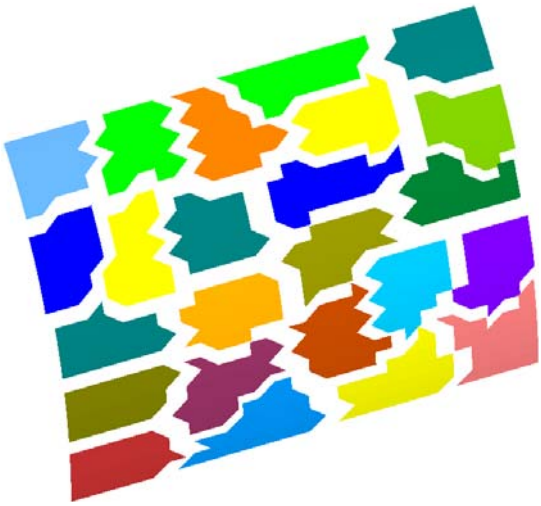

FIGURE 7. Partitioning of 3D shell mesh in 24 subdomains.

- Include different finite elements such as beam, shell, plane 2D, solid 3D;
- Use several parallel solution methods: FETI, primal Schur complement, ...
- Perform static and dynamic analyses.

Postprocessor

- Display mesh deformation and mode shape of structures;
- Display contours, iso-line, iso-surface and vector;
- Display the inside of there dimensional structures by cutting plane;
- Add or cancel light effects;
- Animate the results in any case of display mode;
- Display selected elements group;
- Display time history of multiple data;
- Display the distribution curve of data on a line path;
- Use mouse operation for rotation, pan and zoom as well as node picking.

3. CASE STUDY

In order to validate the solution method and to illustrate the functionalities of FER/SubDomain, the example of cylindrical roof is carried out. In Figure 6 the geometry, the boundary conditions and physical details of the problem are given. The roof is supported by rigid diaphragm at its two ends and has its longitudinal edges free. A linearly distributed force is applied to a symmetrical axe.

For symmetry reason, only a quarter of the roof is modeled by means of triangular shell elements. The mesh, generated by a third-part software, includes 625 nodes and 1152 elements (3750 degrees of freedom). The analysis can be achieved by a series of functions of FER/SubDomain. The mesh is imported in FER/SubDomain and then decomposed into 24 subdomains, as shown in Figure 7.

After the solution, FER/SubDomain undertakes also the task of postprocessing. For instance, the iso-lines of Mises stress of one subdomain, in the case of 3D shell element or of 3D solid element, can be depicted as in Figure 8.

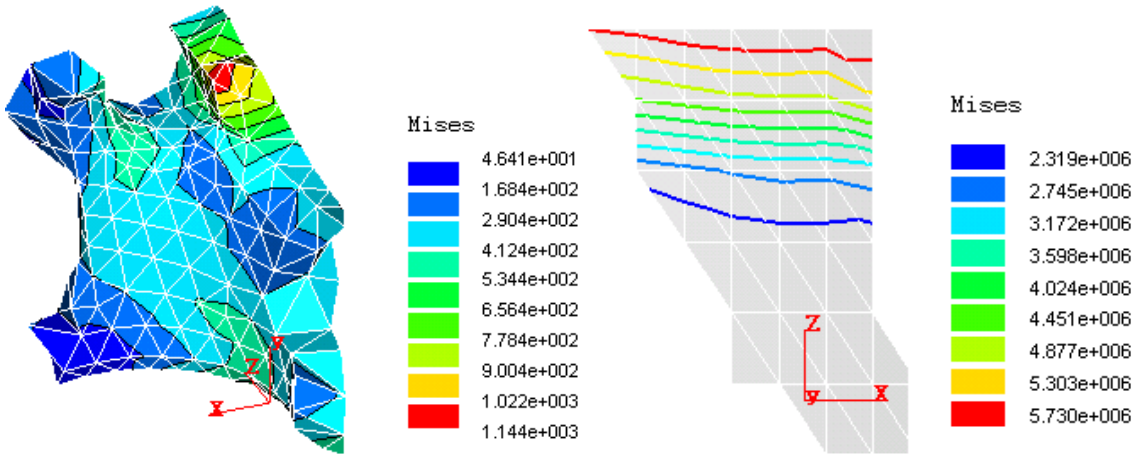

FIGURE 8. Mises stresses in one subdomain.

4. CONCLUSIONS

In this paper, we have presented an practical application of the object-oriented approach to finite element analysis and the product FER/SubDomain. Open architecture of the program facilitates further developments and adapts to suit specific needs easily and quickly. Moreover, the proposed user interface has proved to be satisfactory and flexible. Our experience shows that C++ offers serious benefits for scientific computing. The authors feel confident that object-oriented programming in C++ will promote the development of computational tools for structural analysis and GUI applications.

REFERENCES

[1] K. Aazizou, J. Besson, G. Gailletaux and F. Hourlier, Une approche C++ du calcul par éléments finis, in *Colloque National en Calcul des Structures* **2** (1993) 709–722.
[2] M. Brain and L. Lovette, *Developing professional applications for Windows 95 and NT using MFC*. Prentice Hall PTR (1997).
[3] J.-M. Cros, Parallel modal synthesis methods in structural dynamics. *Contemp. Math.* **218** (1998) 238–245.
[4] J.-M. Cros and F. Léné, Parallel iterative methods to solve large-scale eigenvalue problems in structural dynamics, in *Domain Decomposition Methods in Sciences and Engineering*, P.E. Bjørstad, M. Espedal and D. Keyes Eds., John Wiley & Sons (1997) 318–324. Bergen, Norway, Proceedings from the Ninth International Conference, June (1996).
[5] L.H. De Figueiredo and M. Gattass, A database management system for efficient storage of structural loading. *Comput. & Structures* **32** (1989) 1025–1034.
[6] G. De Saxcé, Le projet charly : un logiciel de calcul par éléments finis et éléments frontières de seconde génération. Séminaire de génie logiciel, Division MSM, Université de Liège (1987).
[7] Y. Duboispelerin, T. Zimmermann and P. Bomme, Object-oriented finite element programming, 2. a prototype program in smalltalk. *Comput. Methods Appl. Mech. Engrg.* **98** (1992) 361–397.
[8] Y. Duboispelerin and T. Zimmermann, Object-oriented finite element programming, 3. an efficient implementation in C++. *Comput. Methods Appl. Mech. Engrg.* **10** (1993) 165–183.
[9] C. Farhat and M. Lesoinne, Mesh partitioning algorithms for the parallel solution of partial differential equations. *Appl. Numer. Math.* **12** (1993) 443–457.
[10] C. Farhat, M. Lesoinne, P. Le Tallec, K. Pierson and D. Rixen, Feti-dp: A dual-primal unified feti method - part i: A faster alternative to the two-level feti method. *Int. J. Numer. Meth. Engrg.* **50** (2001) 1523–1544.
[11] C. Farhat and F.-X. Roux, Implicit parallel processing in structural mechanics, in *Computational Mechanics Advances*, J. Tinsley Oden Ed., Vol. 2, North-Holland (1994) 1–124.
[12] Z.Q. Feng, K. Aazizou and F. Hourlier, Modélisation des problèmes de contact avec frottement implantation en C++ dans le code zébulon, in *Colloque National en Calcul des Structures 2* (1993) 1141–1156.
[13] Z.G. Feng, Z.Q. Feng and M. Domaszewski, Fer/view : un post-processeur général de calcul par éléments finis. Teknea, in *4ème Colloque National en Calcul des Structures 2* (1999) 883–887.

[14] J.S.R.A. Filho and P.R.B. Devloo, Object-oriented programming in scientific computations: The beginning of a new era. *Engrg. Comput.* **8** (1991) 81–87.
[15] B.W.R. Forde, R.O. Foschi and S.F. Stiemer, Object-oriented finite element analysis. *Comput. & Structures* **34** (1990) 355–374.
[16] K.P. Jacobsen, Fully integrated superelements: a database approach to finite element analysis. *Comput. & Structures* **16** (1983) 307–315.
[17] D.L. Kunz and A.S. Hopkins, Structured data in structural analysis software. *Comput. & Structures* **26** (1987) 965–978.
[18] P. Le Tallec, Domain decomposition methods in computational mechanics, in *Computational Mechanics Advances*, J. Tinsley Oden Ed., Vol. 1, North-Holland (1994) 121–220.
[19] P. Le Tallec, J. Mandel and M. Vidrascu, A neumann-neumann domain decomposition algorithm for solving plate and shell problems. *SIAM J. Numer. Math.* **35** (1998) 836–867.
[20] J. Lu, D.W. White, W.F. Chen and H.E. Dunsmore, A matrix class library in C++ for structural engineering computing. *Comput. & Structures* **55** (1995) 95–111.
[21] R.I. Mackie, Object-oriented programming of the finite element method. *Internat. J. Numer. Methods Engrg.* **35** (1992) 425–436.
[22] P. Menetrey and T. Zimmermann, Object-oriented non-linear finite element analysis - application to j2 plasticity. *Comput. & Structures* **49** (1993) 767–777.
[23] T.S Murthy, Y.K Shyy and J.S. Arora, Midas: management of information for design and analysis of systems. *Adv. Eng. Software* **8** (1986) 149–158.
[24] PGSoft and University of Colorado. *TOP/DOMDEC: A totally object oriented program for visualisation, domain decomposition and parallel processing* (1994). User's manual.
[25] R.M.V. Pidaparti and A.V. Hudli, Dynamic analysis of structures using object-oriented techniques. *Comput. & Structures* **10** (1993) 149–156.
[26] S.D. Rajan and M.A. Bhatti, Data management in fem-based optimization software. *Comput. & Structures* **16** (1983) 317–325.
[27] B. Raphael and C.S. Krishnamoorthy, Automating finite element development using object-oriented techniques. *Engrg. Comput.* **10** (1993) 267–278.
[28] S.P. Scholz, Elements of an object-oriented fem++ program in C++. *Comput. & Structures* **43** (1992) 517–529.
[29] H.D. Simon, Partitioning of unstructured problems for parallel processors. *Computing Systems in Engineering* **22** (1991) 135–148.
[30] G. Touzot, S.i.c.1.1: Réflexion sur l'architecture des logiciels de modélisation. Technical report, Université de Technologie de Compiègne (1986).
[31] P. Verpeaux, T. Charras and A. Millard, Castem 2000: une approche moderne du calcul des structures, in *Calcul des structures et intelligence artificielle*, J.M. Fouet, P. Ladevèze and R. Ohayon Eds., Pluraris (1988).
[32] S. Wang, A conception of module library and data base management system for finite element analysis. *Comput. & Structures* **26** (1989) 1073–1083.
[33] R.S. Wright Jr. and M. Sweet, *OpenGL superbible: the complete guide to OpenGL programming for Windows NT and Windows 95*. Waite Group Press (1996).
[34] G.W. Zeglinski, R.P.S. Han and P. Aitchison, Object-oriented matrix classes for use in a finite element code using C++. *Internat. J. Numer. Methods Engrg.* **30** (1994) 3921–3937.
[35] T. Zimmermann, Y. Duboispelerin and P. Bomme, Object-oriented finite element programming, 1. governing principles. *Comput. Methods Appl. Mech. Engrg.* **98** (1992) 291–303.