A GENERAL OBJECT-ORIENTED FRAMEWORK DESIGN FOR NUMERICAL SOLUTION OF POTENTIAL PROBLEMS BY BOUNDARY ELEMENTS

Rogério José Marczak – rato@mecanica.ufrgs.br
Departamento de Engenharia Mecânica - Universidade Federal do Rio Grande do Sul
Rua Sarmento Leite, 425 - Porto Alegre - RS - 90050-170 - Brazil

Abstract. *This work presents an object-oriented architecture to be used as a general numerical framework for the development of computer programs based on the boundary element method (BEM). The proposed design provides a large set of classes developed specifically to handle those entities most commonly found in solution procedures based on boundary or finite elements. The framework is divided into logical units which enable the analyst to "assemble" the code accordingly to the type of problem is to be solved: linear or nonlinear, steady state or transient, type of system of equations, type of solver to be used, type of solution strategy and so forth. The present version is particularized for the solution of potential problems using the BEM. The design allows the use of an arbitrary number of subregions, which are connected automatically by imposing compatibility conditions for potentials and fluxes. This enables the analyst to model problems composed by different material properties or subdomains in a single solution step and, in the case of linear problems, without need of interior meshes. The use of the proposed framework as an included library thus simplify the development cycle of research software for heat flow, torsion, electromagnetic and other problems governed by Poisson equation.*

Keywords: *Boundary Element Method, Object-oriented programming, Potential problems.*

1. INTRODUCTION

During de 90's, the use of object-oriented programming (OOP) has become a common paradigm in most software development fields. The application of OOP has also reflected heavily on engineering software, especially on those commercially available. The diffusion of the use of OO languages in engineering is due to a number of factors. Some of them are directly related to the software development cycle and its corresponding cost. The constant need for software updating and re-releasing was almost reaching its practical limits. In some cases, one can find computer programs that started its development life more than thirty years ago, using a (now considered) obsolete computer language and - not rarely - based on source codes composed by hundreds of thousands of structured programming lines. This naturally has lead to a demand for *extensibility* and *reusability* of the codes (or part of them) without demanding the costs associated to the development of new software or due to unwanted changes in source codes successfully tested and used. This demand is relatively old, but only the born of object-oriented programming (OOP) has been leading to an adequate solution. As a matter of fact, this demand is intimately related to the origins of several OO languages.

This work presents an object-oriented architecture to be used as a general numerical framework for the development of computer programs based on the boundary element method (BEM). The present version is particularized for the solution of steady-state potential problems using an arbitrary number of subregions, thus enabling the analyst to model problems composed by different material / characteristics. The main goal of the present work is to unlink the domain classes (those containing elements, nodes etc.) from the analysis classes (linear, non-linear, steady state, transient etc.).

A detailed description of the OO philosophy will not be covered in this work. It is supposed a basic knowledge in OO programming as well as its fundamental aspects (classes and objects, inheritance and polymorphism). Classes hierarchy and relationship will be illustrated following Rumbaugh (1991) notation.

1.1 Some characteristics commonly found in OO FEM/BEM software

A brief review of the literature reveals many conceptually different possibilities available to develop FEM computer programs using OO philosophy (Archer, 1996; Hedegal, 1994; McKenna, 1997). In the BEM context, the number of published works are much more scarce (Marczak, 1999). Whatever the choice, the essential part of a FEM/BEM software are the solution algorithms of the discretized problem, a determining aspect of the efficiency, robustness, and stability of the solution. This part is generally provided by a set of classes that *govern the flow* of the available data during the solution. In view of the responsibility that these classes have on the solution of the problem, it is expected a higher level of abstraction in their design, and we will refer to them in this work as *analysis classes*.

Another important layer of classes (called *model classes* in this work) is necessary to store and manage the problem data. They *provide the data* of the numerical model to the analysis objects.

On the other hand, good OO designs also provide classes to handle special entities like basic linear algebra objects (vectors, matrices, etc.). These objects *perform auxiliary* (but not less important) *tasks* requested by the analysis and model objects, or help to build higher level objects as an aggregation of them. They will be called *auxiliary classes*.

Accordingly, it seems reasonable to identify three relative levels of abstraction for most classes found in OO FEM/BEM software, as shown in Table 1:

- **Low level**: They implement algorithms that handle tensors, matrices or vectors. A typical example is a class able to solve a linear system or an eigensystem. Although matrix objects are the most common case, other entities like lists (to store the objects), mathematical functions, strings, geometric primitives, memory management, among several others, are also treated in this level.

- **Intermediate level**: These classes generate or store the necessary information for the solution of the problem. These data are responsible for the relationship mathematical problem ↔ physical problem. For instance, it is in this level that one can find FEM/BEM objects like material property, geometry, mesh, numerical integration, boundary condition, superelements and loading.

- **High level**: It is in this level that one or more objects solve the problem from the governing equation point of view. This is accomplished by using and controlling the information generated by objects in the two previous levels. For example, a single mesh object can be used to solve a static linear problem and a dynamic transient problem during the same job. In essence, both problems differ only in the way the high level objects will manipulate the data provided by the intermediate level objects, using the tools provided by the low-level objects.

Table 1. Class category convention adopted in this work.

Level of abstraction	Class category (in a FEM/BEM software)
Low	Auxiliary class
Intermediate	Model class
High	Analysis class

The classification in Table 1 does not mean, for example, that a matrix class actually has a low level of abstraction. But calling it an auxiliary class means (in the context of the present work) that there is not much more that a matrix object can do except for matrix operations. An analysis class, by its turn, can span from a simple linear steady state analysis to a fluid-structure optimization analysis.

2. THE `mcBEM` LIBRARY

The **`mcBEM`** library was started as a research project focusing on applying OO programming to develop flexible, modular, and reusable software components for solving differential equations by the BEM. The underlying idea in its design is that different applications share a common mathematical and numerical structure, and more importantly, storage (model) classes do not perform any solution step. In the form of a compiled library, **`mcBEM`** provides a complete set of auxiliary and model classes, as well as a basic set of analysis classes. To create an application code, the analyst *assembles* the code collecting the objects necessary to perform the solution. The programmer's work is limited to implement the new analysis classes (in case they are not provided), deriving them from any of the existing ones. The objective of this section is to present the basic hierarchy layout of **`mcBEM`** main classes.

2.1 mcBEM model classes

The **mcBEM** model classes are mostly formed by a set of classes derived from a super class called **mcEntity**. They were designed to compose a bulk of entities commonly found in a discrete PDE solution model, like FEM and BEM. But the most important model class of the proposed design is the **mcDomain** class. **McDomain** acts as a container class for the analysis, storing all **mcEntity** objects necessary to describe the problem such as geometry, mesh, loads and boundary conditions. The objects are stored in list objects (**mcList**) especially designed for **mcBEM**, reducing by a significant amount the overhead of general purpose standard lists (like the STL - Standard Template Library Programmer's Guide, 1999).

Figure 1 depicts the basic hierarchy of **mcBEM** model classes. The most important of them will be described in the sequel.

- **mcPoint**: Implements a coordinate point in $\Re^3$ space. It can be attached to a user-defined coordinate system, if desired.
- **mcNode**: Derived from **mcPoint** class, a **mcNode** object represents a point which has degrees of freedom (DOF), i.e. a space location that holds part of the discrete solution for a given mesh.
- **mcCoorSys**: Enable the analyst to use special coordinate systems throughout the solution.
- **mcMaterial**: A class to implement general material properties.
- **mcGeometry**: Implements geometric properties for special applications (like areas in spars, thickness in plates, etc.)
- **mcBESubregion**: Implements a BEM subregion of the solution domain. This can be used to handle problems composed by different materials, geometric properties etc. The **mcBESubregion** class also encapsulates information about the type of the differential equation which is being solved, so that it knows how many DOF's each **mcNode** have, or what are these DOF's. In addition, a **mcBESubregion** object can access the fundamental solution of the problem, and determine whether a DOF is a primal or a dual one. This is necessary to implement compatibility conditions on the interface shared by two or more subregions, as well as to impose the boundary conditions. A similar class - the **mcFESubregion** can be used for finite elements.
- **mcBElement**: This class implements boundary element objects on the boundary of each subregion. The **mcBElement**'s objects are composed by aggregating two super classes: **mcPhysicalPartition** and **mcGeometricPartition**, in such a way that the geometric description of the element may be dissociated from its physical description (Devloo, 1997). For instance, a linear geometric partition can be used along with a constant (one node) physical partition or a linear (two node, continuous or discontinuous) physical partition.
- **mcDCell**: Implements domain cells for problems where an interior mesh is necessary. Similar classes implement finite elements, control volumes etc.
- **mcLoad**: This class is used to create general loads to be applied on the domain. In BEM applications, the boundary conditions are implemented as a special case of loads.
- **mcDomain**: The **mcDomain** class collects and manages all **mcEntity** objects for a given solution domain. That is, it represents the computational model of the problem. It is used by the analysis classes to access all necessary information to solve the discretized problem. Since **mcDomain** objects perform no analysis step, the analyst can solve different domains during the solution phase.

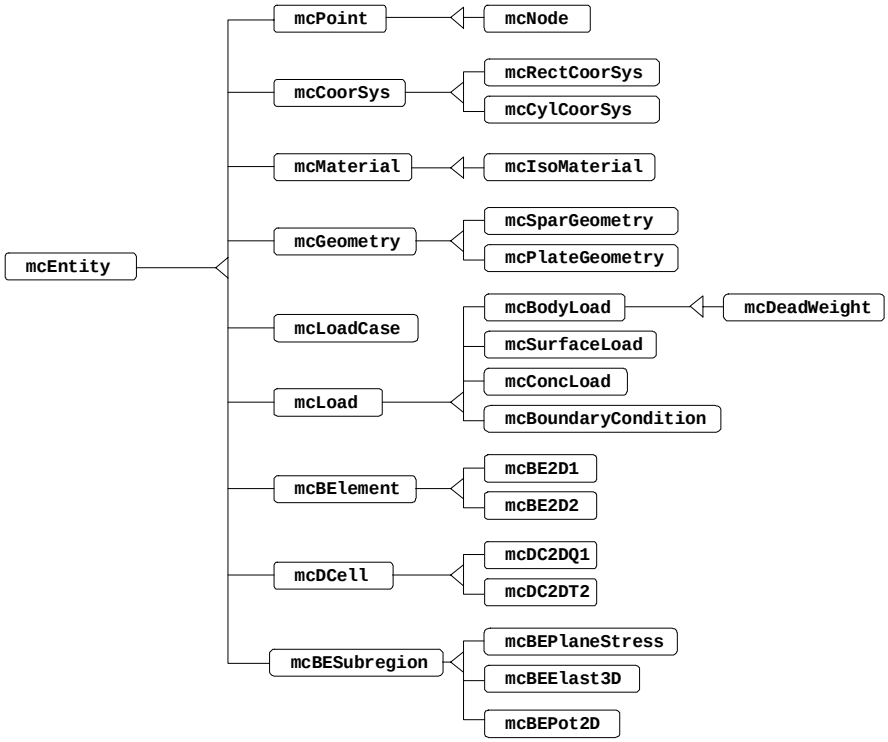

Figure 1: Examples of basic **mcBEM** model classes.

2.2 mcBEM analysis classes

The analysis of the problem is performed by a set of five super classes aggregated by the analyst, depending on the type of the problem and solution desired. Many of the ideas adopted here came or were adapted from the work of McKenna, 1997. This approach adds flexibility by enabling the user to *slot* each one of these five major classes according to the specific needs of each application. If necessary, one can implement a new class by deriving it from any of these five super classes and limiting the coding task to those analysis steps not provided by **mcBEM**. The analysis classes currently implemented in this work are summarized below:

- **mcSolutionAlgorithm**: The **mcSolutionAlgorithm** objects orchestrate the major steps in the analysis. Typical tasks of these objects are: form the left hand side and the right hand side of the linear system, and solve the linear system. In case of linear problems this is generally done only once, but for non-linear problems the steps are repeated until convergence is reached. Currently, two major subclasses are derived from this class: **mcBEMEigenvalueSolAlgo** and **mcBEMEquilibriumSolAlgo**. Both can be particularized for special cases, as illustrated in Fig. 2.
- **mcAssembler**: The **mcAssembler** objects provide methods necessary to form the system of equations. It is responsible for accessing each boundary element, domain cell, finite element or control volume and adding its contributions to the global system of equations. One major subclass is currently derived from **mcAssembler** class: it is the **mcIncrementalAssembler** class, which generates derived classes like **mcStaticAssembler** (for linear steady state problems), **mcTransientAssembler**

(for transient problems) and **mcEigenvalueAssembler** (for eigenvalue problems). Fig. 3 shows the basic hierarchy.

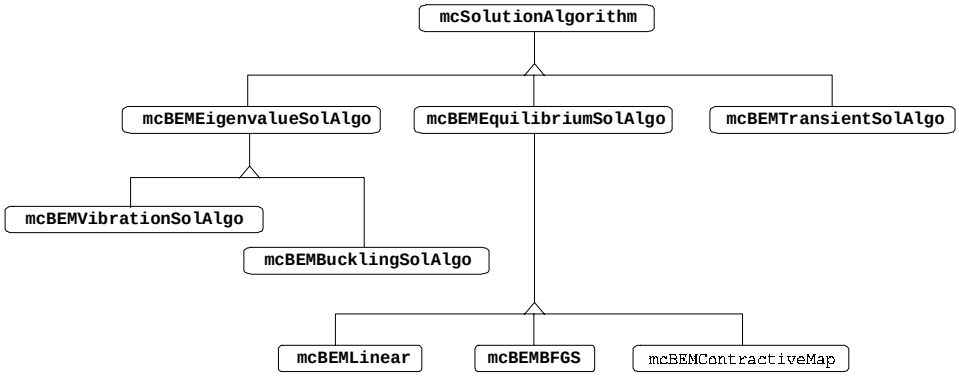

Figure 2: Typical **mcBEM** solution algorithm classes.

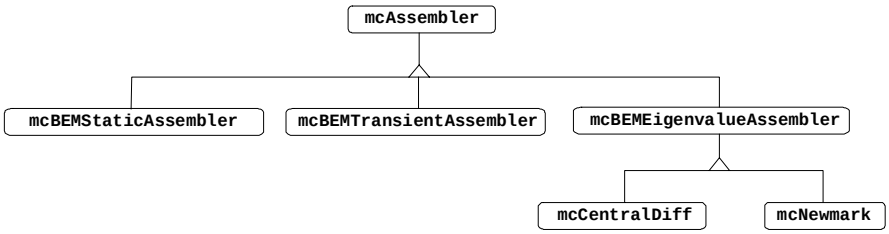

Figure 3: Typical **mcBEM** assembler classes.

- **mcModelHandler**: The model handler objects are responsible for providing access to the objects in **mcDomain** during the solution phase, so that no object in the analysis aggregation needs to access the solution domain directly. Any object in the domain can be reached by **mcModelHandler** methods. Iterators are provided to access nodes, elements etc. Fig. 4 illustrates the basic hierarchy.
- **mcConstraintHandler**: The **mcConstraintHandler** super class implements methods to apply constraints on the system of equations. Prescribed temperatures or fluxes are handled here. The imposition of compatibility conditions on the interface shared by two or more subregions are also handled by **mcConstraintHandler** objects. Other examples of tasks performed here are DOF numbering and Lagrange multipliers handling. In case of the BEM, the **mcConstraintHandler** object is also responsible for automatically creating the nodes of the boundary elements and domain cells..
- **mcAnalysis**: This is the analysis aggregation itself. A **mcAnalysis** object receives all other component objects as arguments. It checks for validity of the aggregation and links them by pointers. A single virtual method: **analyze()** triggers the analysis start up. **mcAnalysis** objects also knows whether the solution domain changed so that is necessary a new analysis (like in adaptive or nonlinear problems) or not. Figure 4 illustrates the subclasses implemented in this work.

2.3 mcBEM auxiliary classes

The **mcBEM** library provides a large number of classes encapsulating many features found in computational mechanics. A few examples are: lists, dictionaries, identification, iterators, error handlers, file handlers, DOF and DOF sets, fundamental solution and numeric integrators.

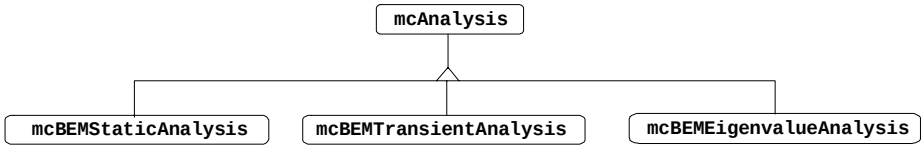

Figure 4: Typical **mcBEM** analysis subclasses.

Of particular importance in the efficiency of a numerical solution is the storage and manipulation of the system matrices that generate the solution of the problem. The present implementation of **mcBEM** provides two major super classes to accomplish this task (see Fig.5):

- **mcSystemOfEquations**: It is responsible for storing the system matrices. It also provides methods to perform the assembly of element sub-matrices as well as rearranging rows and columns to eliminate/add DOF's etc. Several types of storage schemes are provided to accommodate the types of matrices generated by the different methods (banded, full, symmetric, non-symmetric, etc.). The **mcSystemOfEquations** objects do not perform the solution of the system.
- **mcSolver**: This is the super class that actually solves the system of equations. Because it is disconnected from the **mcSystemOfEquations**, a **mcSolver** object can be linked with well-known Fortran solvers or other numerical libraries (Zeglinski *et al.*, 1997).

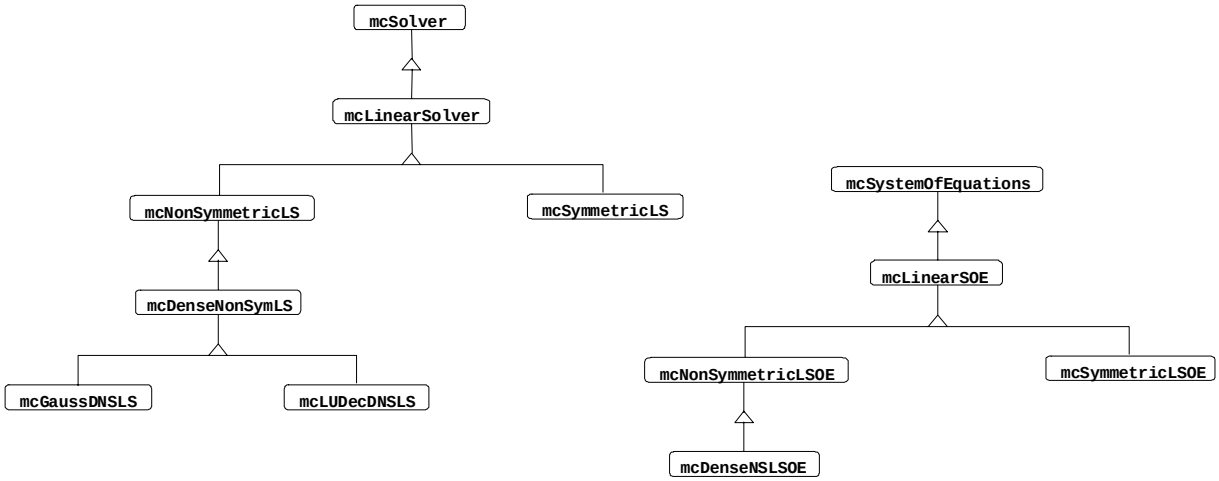

Figure 5: Some of **mcBEM** auxiliary classes.

AN ILLUSTRATIVE EXAMPLE

Suppose the analyst is interested in solving a steady state heat conduction problem like the one depicted in Fig. 6. The primal variables are the temperatures (T) while the dual variables are the normalized heat flux ($q' = q / k$). The domain is composed of two subregions (Ω_1 and Ω_2) of different material properties (k_1 and k_2). The interface Γ_{12} accounts for that part of the boundary of Ω_1 which is shared with Ω_2, and Γ_{21} has an analogous meaning. Let Γ_{12} be formed by boundary elements e_1 and e_2, while Γ_{21} is formed by boundary elements e_7 and e_8.

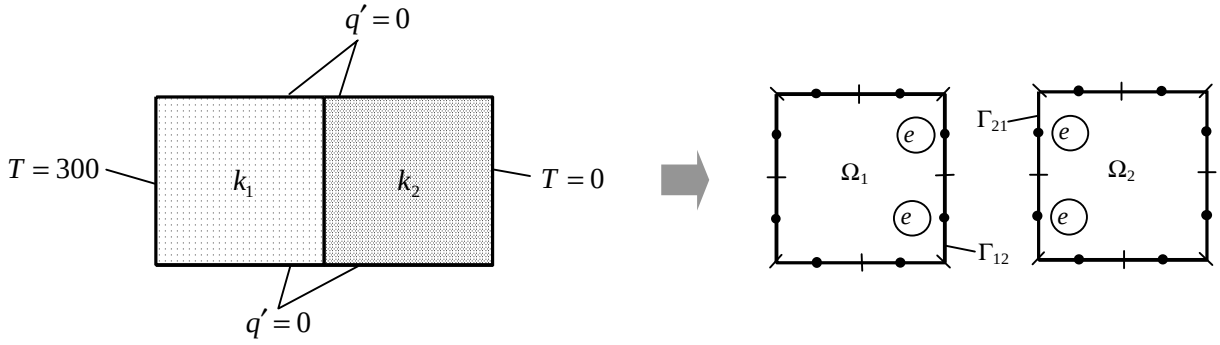

Figure 6: A simple example of domain composed by two subregions.

To solve such problem using the proposed framework becomes very simple by using two **mcPot2D** objects and the two corresponding material properties. As already noted, **mcBEsubregion** objects keep track of the fundamental solutions as well as their DOF's (T and q'). In this case, one could use:

```
1.  mcIsoMaterial       mat_1(k1), mat_2(k2);
2.  mcPot2D             omega_1, omega_2;
```

It is also necessary to define the interface shared by both subregions:

```
3.  mcInterface         interf_12(e1,e2,e7,e8);
```

The code fragment exemplified above suffices to define both materials, both 2D potential subregions, and to enforce the compatibility conditions over the interface, so that $T_{12} = T_{21}$ and $q'_{12} = -q'_{21}$.

A point of further interest is the way the compatibility conditions are imposed. In **mcBEM**, all DOF's are identified by a code number and also by a code name. If two nodes - belonging to different subregions - are interfaced, **mcBEM** compares both names and codes. It they coincide, it is enforced the same values for their primal variables and opposite values for their dual variables. This enables the user to join two subregions governed by different differential equations, provided they share at least one (primal or dual) DOF.

3. REUSABILITY AND EXTENSIBILITY

The main goal of the OO design proposed in this work is to enable the analyst to write a few programming lines to customize the code for a given application. A simple analysis like the one shown in Fig. 6 would be straightforward reusing the relevant objects. Figure 7 shows a possibility. Note that the solution of a linear static 2D structural problem would use the same code (the inherent differences would be hidden in subregion objects properly defined).

If the user is interested in solving the same problem in a transient fashion, it would be necessary to write adequate analysis classes (if they are not provided) and change lines 4, 5 and 10 of the code in Fig. 7 accordingly to Fig. 8.

```
1.  mcDomain                  part_959;

2.  domain.ReadConfigFile("input.dat");
3.  domain.setCurrentLoadCase(domain.getDefaultLoadCase());

4.  mcBEMLinear               theAlgorithm;
5.  mcBEMStaticAssembler      theAssembler;
6.  mcBEMModelHandler         theModel;
7.  mcBEMConstraintHandler    theConstraint;
8.  mcDenseNonSymLS           theSolver;
9.  mcDenseNSLSOE             theSOE(theSolver);
10. mcBEMStaticAnalysis       theProblem( part_959,theAlgorithm,
                                          theAssembler,theModel,
                                          theConstraint,theSOE);
11. theProblem.analyse();
```

Figure 7: A possible code fragment for a steady state analysis of the problem in Fig.6.

```
1.  mcDomain                  part_959;

2.  domain.ReadConfigFile("input.dat");
3.  domain.setCurrentLoadCase(domain.getDefaultLoadCase());

4.  mcBEMTransientSolAlgo     theAlgorithm;
5.  mcBEMTransientAssembler   theAssembler;
6.  mcBEMModelHandler         theModel;
7.  mcBEMConstraintHandler    theConstraint;
8.  mcDenseNonSymLS           theSolver;
9.  mcDenseNSLSOE             theSOE(theSolver);
10. mcBEMTransientAnalysis    theProblem( part_959,theAlgorithm,
                                          theAssembler,theModel,
                                          theConstraint,theSOE);
11. theProblem.analyse();
```

Figure 8: A possible code fragment for a transient analysis of the problem in Fig.6.

4. CONCLUSIONS

This work presented a modular, reusable and extensible OO design for numerical solution of potential problems using the BEM. The main super classes of the proposed architecture were presented, showing the independent role that storage and solution classes play. This enable the user to customize the code for a given application by modifying only a few lines of the driver program.

REFERENCES

Archer, G.C., 1996, Object-Oriented Finite Element Analysis. PhD thesis, University of California at Berkeley.

Beck, R., Erdmann, B. & Roitzsch, R., 1995, Kaskade 3.0 - an object-oriented adaptive finite element code. Technical report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, TR 95-4.

Besson, J., Foerch, R., Cailletaut, G., Aazizou, K. & F. Hourlier, F., 1997, Large scale object oriented finite element code design. Preprint.

Devloo, P.R.B., 1997, PZ: An object oriented environment for scientific programming. Comput. Methods Appl. Mech. Engrg., 150, 133-153.

Dubois-Pèlerin, Y. & Zimmermann, T., 1993, Object-oriented finite element programming: III. an efficient implementation in C++. Comput. Methods App. Mech. Engrg., 108, 165-183.

Feijóo, R.A., Guimarães A.C.S. & Fancello, E.A., 1991, Some experiences with object-oriented programming and their applications in the finite element method (in Spanish). LNCC - Laboratório Nacional de Computação Científica - Brazil, Report 015/91.

Forde, B.W.R., Foschi, R.O. & Steimer, S.F., 1990, Object-oriented finite element analysis.Computers & Structures, 34(3), 355-374.

Hedegal, O., 1994, Object-Oriented Structuring of Finite Elements. PhD thesis, Aalborg University.

Kong, X.A. & Chen, D.P., 1995, An object-oriented design of fem programs. Computers & Structures, 57(1), 157-166.

Langtangen, H.P., 1996, Details of finite element programming in diffpack. Technical report, Department of Mathematics, University of Oslo.

Lu, J., White, D.W., Chen, W.F. & Dunsmore, H.E., 1995, A matrix class library in C++ for structural engineering computing. Computers & Structures, 55(1), 95-111.

Mackie, R.I., 1992, Object oriented programming of the finite element method. Int. J. Num. Meth. Engng., 35, 425-436.

Marczak, R.J., 1999, A partial review of object-oriented architectures for finite element programs (in Portuguese), in: Proc. 15th Brazilian Congress of Mechanical Engineering, Águas de Lindóia, São Paulo, Brazil.

McKenna, F.T., 1997, Object-Oriented Finite Element Programming: Frameworks for Analysis, Algorithms and Parallel Computing. PhD thesis, University of California at Berkeley.

Menétrey, P. & Zimmermann, T., 1993, Object-oriented non-linear finite element analysis: Application to J2 plasticity. Computers & Structures, 49(5), 767-777.

Miller, G.R., 1991, An object-oriented approach to structural analysis and design. Computers & Structures, 40(1), 75-82.

Olsson, A., 1998, An object-oriented implementation of structural path-following. Comput. Methods Appl. Mech. Engrg., 161, 19-47.

Pidaparti, R.M.V. & Hudli, A.V., 1993, Dynamic analysis of structures using object-oriented techniques. Computers & Structures, 49(1), 149-156.

Rumbaugh, J., Blaha, M., Premerhani, W., Eddy, F. & W. Lorensen, W.,1991, Object-Oriented Modeling and Design. Prentice-Hall.

Scholz, S.P.,1992, Elements of an object-oriented FEM++ program in C++. Computers & Structures, 43(3), 517-529.

Standard Template Library Programmer's Guide, 1999, Silicon Graphics Computer Systems, Inc.

Zeglinski, G.W., Han, R.P.S. & Aitchison, P., 1994, Object-oriented matrix classes for use in a finite element code using C++. Int. J. Num. Meth. Engng., 37, 3921-3937.

Zimmermann, T. & Dubois-Pèlerin, Y. & Bomme, P., 1992, Object-oriented finite element programming: I. governing principles. Comput. Methods App. Mech. Engrg., 98, 291-303.