Object-Oriented Implementation of an Interactive and Automatic Field-Processing Surface

Herbert De Gersem, Uwe Pahner, and Kay Hameyer

***Abstract*—Post-processing of multiple and hybrid finite element field solutions requires a post-processor that is more general than standard available tools. A novel three-level hierarchic post-processor structure is proposed. The first level of the post-processing environment is a library of programming objects representing mathematical entities such as fields, tables, geometries, meshes and numbers and of operations manipulating them. Frequently used post-processing tasks may be coded explicitly in this kernel and compiled into executable code. An interpreter for mathematical expressions forms the second level of the post-processor structure. Characteristic sequences of operations can be gathered in scripts or in functions and interpreted at run-time. The third level passes commands to the field parser or to the visualization routines. Selected simulation examples demonstrate the capabilities of the chosen approach.**

***Index Terms*—Design automation, finite element methods, object oriented programming, software libraries.**

I. Introduction

SINGLE field finite element processing is commonly used in computer aided design [1], computer assisted teaching [2] and numerical analysis. Standard post-processing tools to obtain derived information out of field solutions are available. The design and research in electric and magnetic fields deal with coupled field problems [3] and automated iterative design procedures [4]. As a result, the complexity of the post-processing schemes increases. A choice has to be made between generally applicable post-processors and easy to use post-processing tools. The question arises if the desired properties of generality and convenience can be brought in agreement. In this paper, it will be shown that a novel object-oriented implementation of an entire post-processing environment is capable to meet such requirements.

II. Post-Processor Versus Post-Processing Environment

Here, a distinction is made between a single post-processor and a post-processing environment. A post-processor provides the indispensable interface between the user and the finite element solution for a field problem [1]. A lot of numerical field computations result in a potential field that is different from the quantities of interest. As a result, a post-processing step is required to derive the values of information from the more abstract finite element solution. Engineers interprete the magnetic behavior of a device by a plot of the magnetic flux lines or the magnetic flux density distribution rather then using a plot of the magnetic vector potential distribution. The aim of the field computation can also be a global quantity such as an inductance or a force [4]. The post-processor performs a role as a buffer between the abstract level of the finite element software and the engineering world of magnetic fluxes, temperatures and displacements. The most important features of a post-processor are its user-friendliness and its clarity. Therefore, most post-processors are developed for single field processing and for fields with one specific nature and discretization. The number of possible

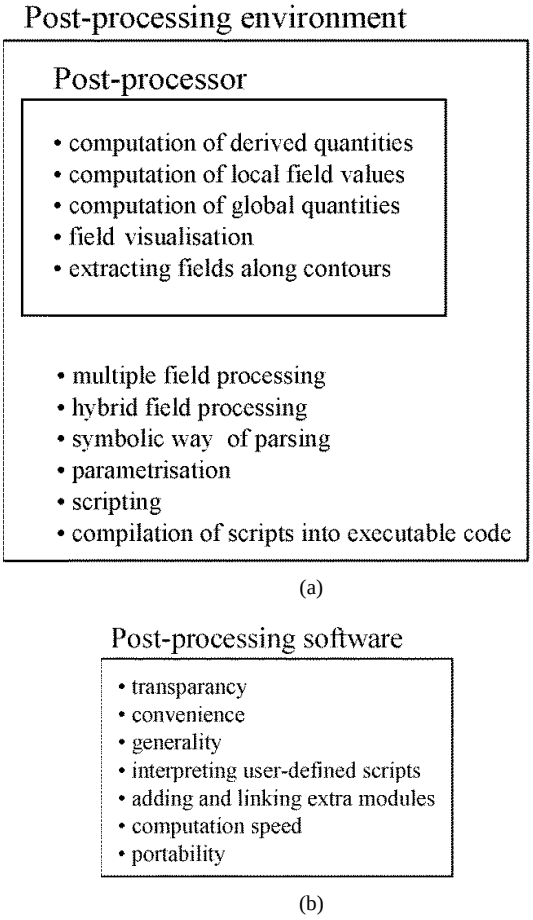


Fig. 1. (a) Features of a post-processor and a post-processing environment; (b) Properties of post-processing software.

Manuscript received October 25, 1999. This work was supported by the Belgian "Fonds voor Wetenschappelijk Onderzoek Vlaanderen" (project G.0427.98) and the Belgian Ministry of Scientific Research, IUAP no. P4/20 on Coupled Problems in Electromagnetic Systems. The research Council of the K.U. Leuven supports the basic numerical research.

The authors are with the Katholieke Universiteit Leuven, Dep. ESAT, Div. ELEN, Kardinaal Mercierlaan 94, B-3001 Leuven, Belgium (e-mail: {Herbert.DeGersem; Uwe.Pahner; Kay.Hameyer}@esat.kuleuven.ac.be).

Publisher Item Identifier S 0018-9464(00)04960-8.

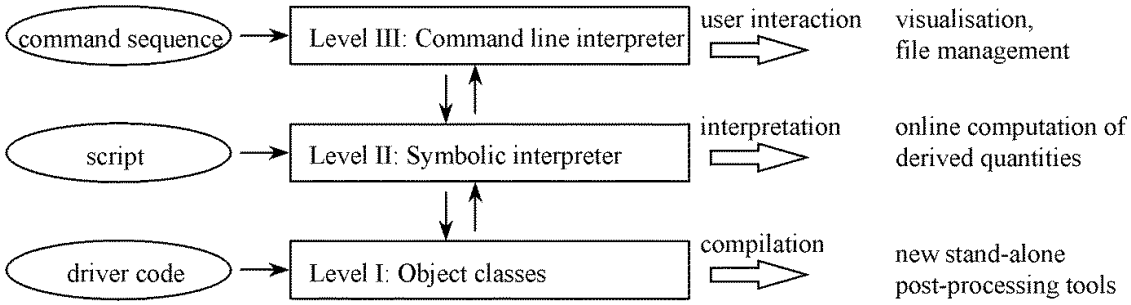


Fig. 2. Hierarchical structure of the post-processing environment.

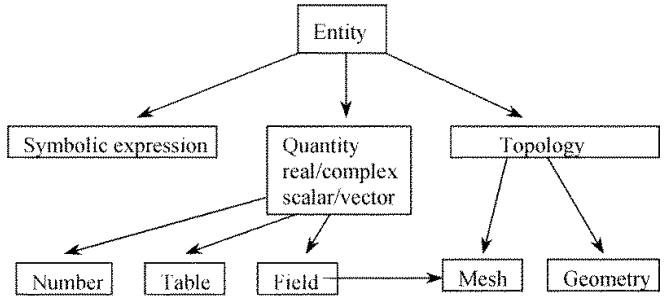


Fig. 3. Class hierarchy of the different entities.

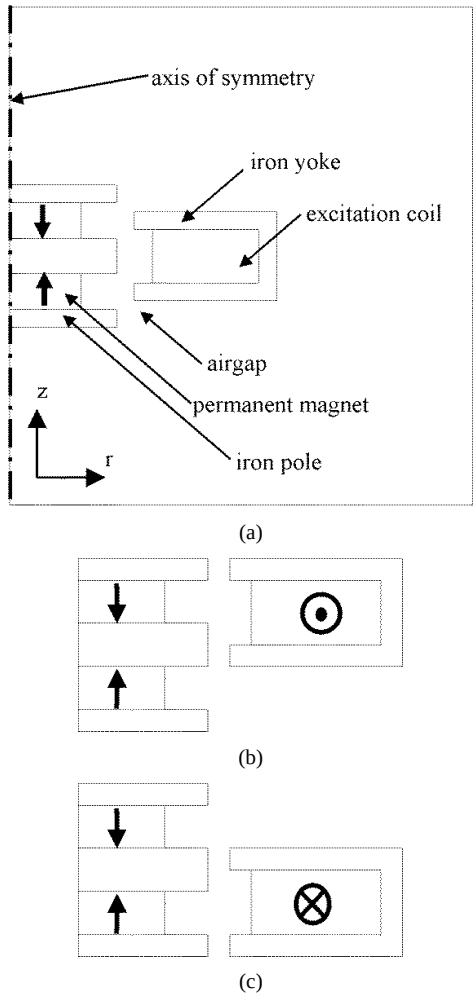


Fig. 4. (a) Simulation example of a magnetic selector; stable position of the magnetic selector (b) for negative excitation and (c) for positive current excitation.

post-processing operations is often limited and forbidden operations are excluded.

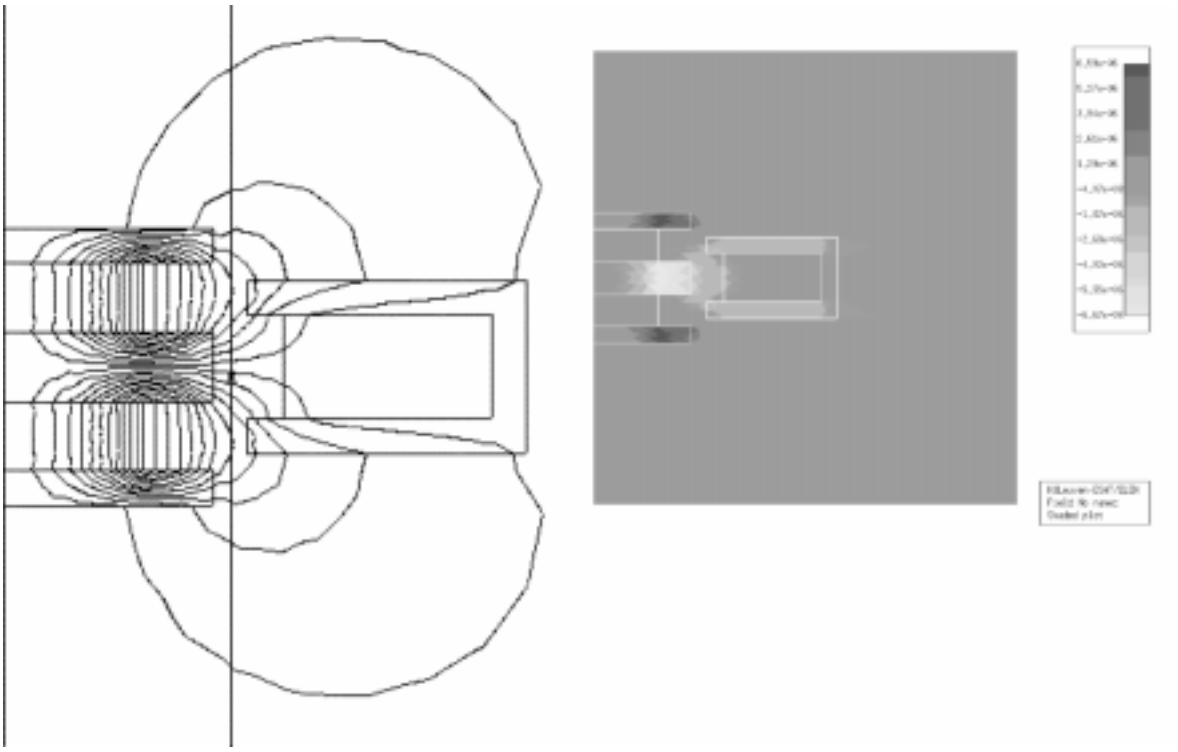
A post-processing environment on the other hand, incorporates and couples fields of different natures and with hybrid discretization techniques. It is possible to adapt the environment to future requirements. It is also intended to use parts of the post-processing environment to build problem specific post-processors. For research and advanced engineering purposes, an interactive and scripted use of the environment is offered. It is difficult to provide a clear, general and robust syntax and sufficiently protect against unacceptable use of post-processing commands. The high number and the abstract formalism of the commands, require an expert to work with the environment. Fig. 1(a) shows the additional post-processing features of the general environment compared to those of the single tool. The properties of a good post-processing software are pointed out in Fig. 1(b).

III. Object-Oriented Design

The development of a software that is both, general and transparent, requires a modular approach [5]. Modularity in operations is achieved by splitting up complex operations into combinations of simpler ones. This requires a top-down structure of the software design. However, one of the major difficulties in a general post-processor is the handling of a huge amount of data of different origine and nature. Fields and numbers have their own storage protocols. Modularity for data treatment is maintained by using a data-driven programming philosophy. Additionally, operations may operate on data independently of its nature, may be specific for one kind of data or may combine different forms of data into a new one. Modularity on the level of data and their characteristic operations, is obtained by choosing an object-oriented approach. The underlying concept of object-oriented design is that one should treat both, data and operation in the software system, as collections of co-operating objects within a hierachy of classes [6]. This means that a link is made between the top-down structured and data-driven design methods. Data are gathered in objects. Operations working upon a single as well as upon multiple objects are defined as member or friend functions associated with the appropriate objects. Progress in execution is obtained as state changes of objects due to functions operating on them. The implementation uses the higher-order C++ programming language [5], [7].

```
phi=read('SOLU.TXT');          % potential solution
r=x(phi);                      % r-coordinates
A=r*phi;                       % magnetic vector potential
Br=-ddy(A);                    % r-component magn flux dens
Bz=ddx(phi)/r;                 % z-component magn flux dens
cv=line(3.25e-3,-5e-3,3.25e-3,9e-3);       % contour
tens=NU0*Br*Bz*2*PI*r;         % Maxwell stress tensor
fcv=tens/cv;                   % restriction to the contour
Fz=inte(fcv);                  % integration
plot(phi,'e24');               % 24 flux lines (Fig.6a)
plot(cv);                      % contour plot (Fig.6a)
plot(Br,'sh');                 % shaded plot Br (Fig.6b)
plot(Br/cv);                   % contour plot Br (Fig.7a)
```

Fig. 5. Interactive commands for the post-processing environment.



(a) (b)

Fig. 6. (a) Flux line plot of the magnetic field with the curve for field evaluation and (b) r-component of the magnetic flux density in the magnetic selector.

The conceptual properties of object-oriented design, especially attractive for developing a post-processing environment, are

1) *Data hiding:* As some entities in the post-processor require a large amount of data, gathering data in structures is extremely useful. A mesh is easily told to be a collection of triangles but consists of several lists representing the elements, the nodes and the mesh connectivity. The construction of a mesh class hides the practical data manipulating from the higher level programming.
2) *Member functions:* Functions can be associated with the data upon which they operate. A function rotating parts of a mesh for instance becomes a member function of the mesh class.
3) *Abstraction and inheritance:* Fields, tables, numbers and symbolic expressions are mathematical quantities upon which mathematical operations are defined. These classes share a common part of data and functions. Meshes and geometries have a topological nature. Translation and rotation are member functions of a common class representing the topology. At last, all classes are entities and may be represented by a name, treated by symbolic expressions and stored in memory or on disk.
4) *Function overloading:* A lot of operations have the same nature but different practical implications on different objects. The addition of numbers and fields are somehow related in the mathematical sense but differ from the low-level programming point of view. Function overloading enable the abstraction of operations.

Additionally, standard building blocks such as single/double linked lists, dictionaries and handle classes, are used to overcome problems of data storage, data indexing, multiple referred objects and memory management [7].

IV. Hierarchical Structure

To satisfy the required but contradictory properties of generality and transparancy, the post-processing environment is built as a three-level hierarchy (Fig. 2). As top-down principle, the levels represent an increasing abstraction from the concept of the physical field to the concept of the finite element solution. The user is invited to interact with the environment at the level that is most suited for his application. Visualization and online post-processing tasks are performed in an interactive top level. Problem specific shells, repetitive tasks and mathematical field computations operate on the second level. Frequently used routines, problem dependent post-processing tools and new finite element applications arise from specific compilations using the building blocks of the lowest level.

A. Level I: Constitutive Post-Processing Objects

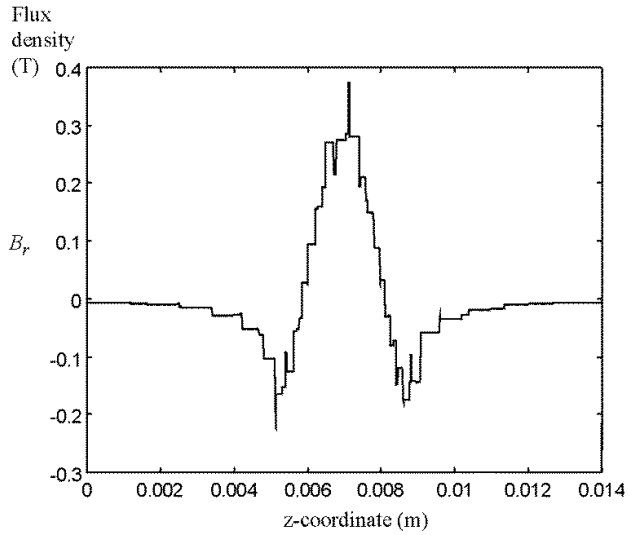
The mathematic offers a general and robust framework to express and simulate the physical reality and its behavior. In a first step all present quantities are defined. Fields, meshes, values, tables, geometries and symbolic expressions of them are characterized by their corresponding objects (Fig. 3). All entities may additionally be complex quantities or vectors. In the proposed post-processing environment, all fields have a discrete nature, they are solutions on a certain discretization of the domain. Tables are lists of data samples. In a second step, all operations operating on single as well as multiple objects, are defined. A mathematical syntax combined with a programming language, capable of overloading functions and operations, provide a natural mechanism for this.

The library of objects and operations can be shared between the finite element solvers and the post-processing environment. Common objects ensure the compability between the finite element solvers and the post-processing tools. For instance, new features for meshes will be automatically transferred to the post-processing environment if both applications share the same mesh object.

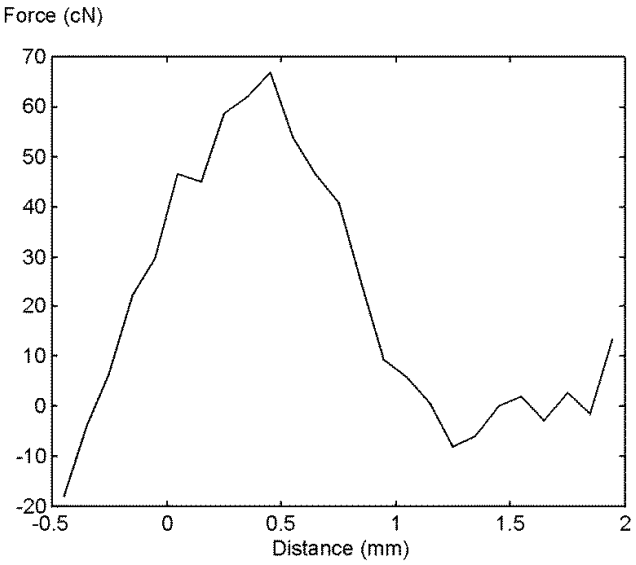
At this stage, the post-processing environment has become a box of bricks for coding finite element computations. The classes hide the low level data and operations of the objects from unauthorised use. Rather circumstantial operations on fields as for instance the multiplication of two fields f_1 and f_2 with different meshes and/or elements of various polynomial degrees, become simply expressable as "f1 $*$ f2."

B. Level II: Symbolic Parser for Finite Element Post-Processing

If multiple fields, curves, values and meshes are involved, the post-processor requires either a stack structure or a symbolic driver, able to distinct and combine the different entities [1]. A stack oriented post-processor puts all entities on a stack. Stack operations shuffle the stack to present the right entities to the calculation engine. For entities with a different nature, multiple



(a)



(b)

Fig. 7. (a) B_r along a line in the middle of the airgap of the magnetic selector and (b) force acting on the translator as a function of its position relative to the excitation coil of the magnetic selector.

stacks can be set up. Organizing a stack can be rather clumsy for the user. Therefore, here, a choice is made for the more mathematically related symbolic way of entity management. A unique name is associated with each entity. A new entity is built by a symbolic expression in terms of the names of yet existing entities. As long as the data itself is not required, the new entity only exists as a symbolic expression. When the entity has to be reported, visualised or stored, the symbolic expression is parsed and the received data replaces the symbolic expression associated with the user-defined name. This approach of postponed calculation prevents the computation of entities that are not further used or visualised.

As some post-processing tasks are characteristic and frequently used, the possibility to script sequences of operations is provided by this software implementation. This is an easy way to extend the effectiveness of the general post-processing environment to problem specific shells and user-defined

```
// open solution file and load solution
printf("Name of input file        : %s\n",solufilename);
MSPField phi(solufilename);

// calculate Maxwell stress tensor
MSPField r=ccx(phi);
MSPField dphidr=ddx(phi);
MSPField dphidz=ddy(phi);
MSPField tensr=(dphidz*dphidz-dphidr*dphidr)/r/2.0;
MSPField tensz=dphidr*dphidz/r;

// read contour file and load contour
printf("Name of contour file   : %s\n",contourfilename);
Curve cv(contourfilename);

// force calculation
MSPTable tabr=cv.extract(tensr);
MSPTable tabz=cv.extract(tensz);
MSPTable hill=cv.hilltable();
double Fz=-2*PI*NU0*inte(tabr*cos(hill)+tabz*sin(hill));

// final output
printf("\nTotal Fz                  : %13.6le N\n\n",Fz);
```

Fig. 8. C++ engine for axisymmetric force computation using the Maxwell stress tensor.

post-processing routines. Functions with their associated parameters and return values are used as a formalism.

The syntax of the expressions have the same mathematical nature as the operations defined in level I. This enables the transfer of the script from the second to the first level, its compilation into fast executable code and/or its addition as a module to the post-processing environment. New post-processing features may originate from user interactions and scripting. As soon as they are mature, the script evaluates to a proper part of the post-processing environment.

C. Level III: Command Line Interpreter

The topmost level provides aside the symbolic parser an interactive way for visualization. It ensures the interface to other software.

V. Application

The developed post-processing environment is used for the post-processing of an axisymmetric magnetic model. A magnetic selector is excited by a coil embedded in an iron C-core (Fig. 4). The moving part consists of two inversely magnetised permanent magnets separated by iron yokes. Depending on the sign of the excitation current, one of both possible stable positions is chosen (Fig. 4). The magnetic field is computed using the $\phi = rA_\theta$ potential in the axisymmetric differential equation [8], [9].

$$\frac{\partial}{\partial r}\left(\frac{v}{r}\frac{\partial \phi}{\partial r}\right) + \frac{\partial}{\partial z}\left(\frac{v}{r}\frac{\partial \phi}{\partial z}\right) = -J_\theta. \tag{1}$$

The magnetic flux density is

$$(B_r, B_z) = \left(-\frac{\partial A_\theta}{\partial z}, \frac{1}{r}\frac{\partial (rA_\theta)}{\partial r}\right) = \left(-\frac{1}{r}\frac{\partial \phi}{\partial z}, \frac{1}{r}\frac{\partial \phi}{\partial r}\right). \tag{2}$$

The force is computed using the Maxwell stress tensor [1]. For axisymmetric problems, the force components in the θ-direction and the r-direction vanish. The force in the z-direction is given by

$$F_z = \int_{z_1}^{z_2} v_0 B_r B_z 2\pi r \, dz \tag{3}$$

along a line parallel to the axis of symmetry and

$$F_z = \int_{r_1}^{F_2} \frac{v_0}{2} (B_z^2 + B_r^2) 2\pi r \, dr \tag{4}$$

along a line perpendicular to the axis of symmetry.

Fig. 5 shows the interactive post-processing commands and parameters for the axisymmetric magnetic problem. The syntax uses operation and function calls. Partial differentiation is performed using the functions "ddx( )" and "ddy( )." A field is evaluated on a contour using the "/"-operator. The post-processor includes the definition of a contour, the computation of the magnetic flux density using (2), the calculation of the force in the z-direction using (3) and the visualization of the fields and their restrictions (Figs. 6 and 7(a)).

As force computation is technically important and frequently used, a stand-alone C++ routine is designed and compiled (Fig. 8). The force calculation tool uses a general contour that may consist of several successive primitives, arbitrarily oriented. Nevertheless, the syntax remains the same as on the interactive level. In the example, the compiled executable is used for the quick, automated and parametrised computation of the force as a function of the position of the translator relative to the excitation part of the magnetic selector (Fig. 7(b)).

VI. Conclusions

With the recent progress in coupled problem modeling and hybrid field discretization techniques, suitable post-processing

tools are recommended. The development of a general post-processing environment which is able to process both multiple and hybrid fields and extendable to future requirements, attracts particular attention to the concept and to the programming techniques. Here, the choice for a three-level structure enables adequate interaction with the environment for both normal skilled users and researchers. Object-oriented programming techniques such as data and function hiding, inheritance and function overloading, create a fire-wall between the low-level experimental coding that is permanently under construction, the mathematical parser, providing the general field processing, and the user interface for convenient post-processing. The three-level hierarchy supports also the automating features. The mathematical parser interpretes scripts. By using the programming objects of the lowest level, scripts may be compiled and added to the post-processing environment. Simulations of a technical example, a magnetic selector, demonstrate the suitability of the developed object-oriented approach. The described post-processor is succesfully implemented in the in-house software package Olympos2D.

REFERENCES

[1] D. A. Lowther and P. P. Silvester, *Computer-Aided Design in Magnetics*. Berlin and New York: Springer-Verlag, 1985.
[2] K. Hameyer, R. Belmans, R. Hanitsch, and R. M. Stephan, "CAT: Computer assisted teaching in magnetics," *IEEE Trans. Magn.*, vol. 34, no. 5, pp. 3304–3307, 1998.
[3] K. Hameyer, J. Driesen, H. De Gersem, and R. Belmans, "Computation of quasistatic electromagnetic fields with respect to coupled problems," in *Proc. of the 8th Int. IGTE Symp. on Numerical Field Calculation in Electrical Engineering*, 1998, pp. 100–105.
[4] U. Pahner, R. Mertens, H. De Gersem, R. Belmans, and K. Hameyer, "A parametric finite element environment tuned for numerical optimization," *IEEE Trans. Magn.*, vol. 34, no. 5, pp. 2936–2939, 1998.
[5] B. Stroustrup, *The C++ Programming Language*, 2nd ed. Reading: Addison-Wesley Publishing Company, 1992.
[6] G. Booch, *Object-Oriented Analysis and Design*. Redwood City: The Benjamin/Cummings Publishing Company, 1994.
[7] B. Sroustrup, *The Design and Evolution of C++*. Reading: Addison-Wesley Publishing Company, 1994.
[8] K. J. Binns, P. J. Lawrenson, and C. W. Trowbridge, *The Analytical and Numerical Solution of Electric and Magnetic Fields*. Chichester: Wiley, 1994.
[9] P. P. Silvester and R. L. Ferrari, *Finite Elements for Electrical Engineers*, 2nd ed. Cambridge: Cambridge University Press, 1990.