iMOOSE—An Open-Source Environment for Finite-Element Calculations

D. van Riesen, C. Monzel, C. Kaehler, C. Schlensok, and G. Henneberger

***Abstract*—Implementing finite-element (FE) solvers for new formulations is often a tedious task, as many common parts are coded again and again. Also, commercial codes are often expensive and therefore difficult to include in, e.g., a teaching environment. iMOOSE is an open-source software package for FE calculations that tries to solve these issues. It is a general-purpose class library that allows for an easy implementation of new FE solvers or FE-related tools. Also included are ready-to-use solvers for electromagnetic calculations and a powerful post-processing tool. Due to its open-source nature, the source code can be examined, modified, and extended to fit the user's needs.**

***Index Terms*—Finite elements, object-oriented software, open-source software.**

I. Introduction

IMOOSE [1] is an open-source environment for finite-element (FE) calculations. It has been developed over the past years out of the necessity of redesigning and re-implementing the same procedures and data structures over and over again when a new solver or formulation was developed. Using iMOOSE as an object-oriented class library, it is now possible to reuse all the basic building blocks of an FEM program.

The class library includes element classes for different two-dimensional and three-dimensional (2-D and 3-D) element shapes and types, problem-definition classes, and equation-solving methods. For specialized tasks, interfacing to external libraries is implemented.

Object-oriented software development methods have acquired some popularity in the last years, and the finite element method (FEM) is well suited for this kind of technique. Different approaches are described in [2]–[4]. In contrast to some more academic decompositions of the FE procedure from the mathematical point of view, iMOOSE takes a more practical approach, also paying attention to the usability of the class library for practical applications and to producing performant solvers, capable of handling large problems [5].

Using the provided class library, a number of solvers for electromagnetic, structural-dynamic, and thermal computations have been implemented. Of general interest, and therefore included in the open-source release, are the solvers for 2-D static and transient and 3-D static and time-harmonic electromagnetic calculations.

The open-source idea is considered by the authors to be a very important contribution to the development and distribution of software. Most of the external libraries, e.g., for equation solving or mesh partitioning used by iMOOSE, are open-source products. iMOOSE has been also released under an open-source license. This allows other users in the scientific community to use the provided programs, review the design and implementation of the library, contribute new functionalities, or use it as a base for their own programs.

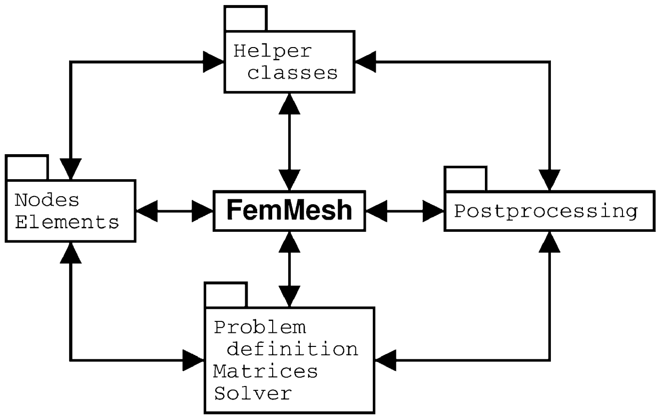

Fig. 1. Overview of the iMOOSE class library.

This paper will give an overview of the class library and present the released solvers. The example of parallel execution of a 3-D static electromagnetic solver will demonstrate how some functionality is delegated to external libraries. Due to this use of external libraries, there are some prerequisites for compiling and running iMOOSE, which are also discussed.

Since iMOOSE is an open project, contributions are possible from sources foreign to the core developer team. This, and the fact that there are different developers working on this project, makes it necessary to implement some coding rules and revision control, as explained in the last section of this paper.

II. Class Library Design

The class library is divided in four main areas (Fig. 1). One is a complex comprising nodes and elements. The other parts include a problem definition and solving area, where system matrices are built and equation systems are solved. For the post-processing of the solution, classes representing physical fields and quantities are available. Helper classes, which, e.g., interface with external mesh generation tools, complete the class library.

The basic class diagram in UML notation [6] of the element classes is shown in Fig. 2.

Implemented element shapes include 2-D triangles and quads, and 3-D tetrahedra, pyramids, prisms and brick el-

Manuscript received July 1, 2003.

The authors are with the Department of Electrical Machines (IEM), Institute of Technology Aachen (RWTH), D-52056 Aachen, Germany (e-mail: riesen@iem.rwth-aachen.de).

Digital Object Identifier 10.1109/TMAG.2004.825471

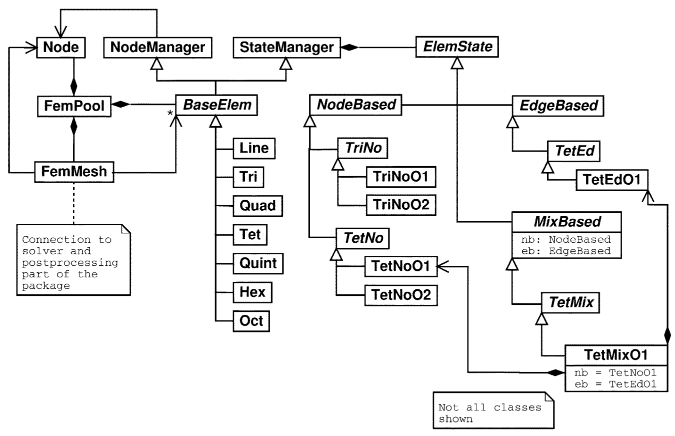

Fig. 2. Basic class diagram of the element classes.

ements. For node-based approaches both first and second order elements are available. Edge-based approaches use only first-order tetrahedra at the moment. Through heavy reuse of this code, the possibility to mix edge- and node-based formulations is also achieved.

In the class diagram, two different class trees can be seen. The basic element (class BaseElem), which represents the only interface to element functions for the rest of the class library, has specializations depending on the geometrical shape of the element. In these, functions not depending on the solver type are implemented, like, e.g., the gauss points and weights for numerical integration. Solver-dependent functions like the shape functions, interpolation of the field quantities in the element, or the building of element matrices and vectors is delegated to the class tree starting with ElemState. Here, a distinction is made between node-based and edge-based elements and first- or second-order elements. For calculations including mixed-type elements with both edge-based vector quantities and node-based scalar quantities, a third element state is defined. To avoid code duplication, this element state class holds pointers to corresponding node- and edge-based classes and only re-implements new functionalities like mixed-element matrices. Thus, the effort needed for the introduction of these mixed elements is very low.

The elements and nodes are linked to the other parts via the concept of a finite-element mesh. Multiple meshes may coexist in the same program.

Another important part is the problem formulation and solving task. The main components can be seen in the class diagram in Fig. 3.

The core class is FemProbBase, which provides an interface to the system matrix and the vector of unknowns and the right-hand-side vector through the use of an equation system (EqunArray). Translation between the id of an unknown (node or edge and component) and the row in the matrix is done by an interface class, with specializations for single degree of freedom or multiple degrees of freedom cases. Extra functionality for multiple degrees of freedom and extra blocks in the system matrix (e.g., for coupling with external circuit equations) is provided by FPMultDof and FPExtraBlocks, respectively. The formulation of an FE problem is defined in the method buildMatrix(). This is the method to be implemented for a new formulation when extending the class

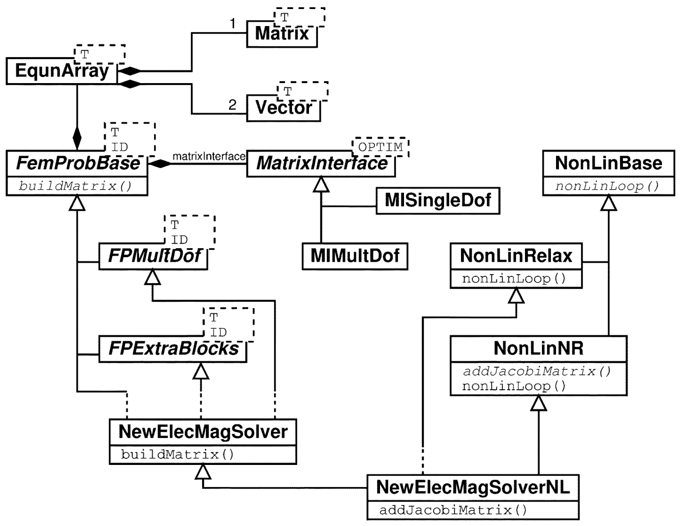

Fig. 3. Class diagram showing the class defining and solving FE problems.

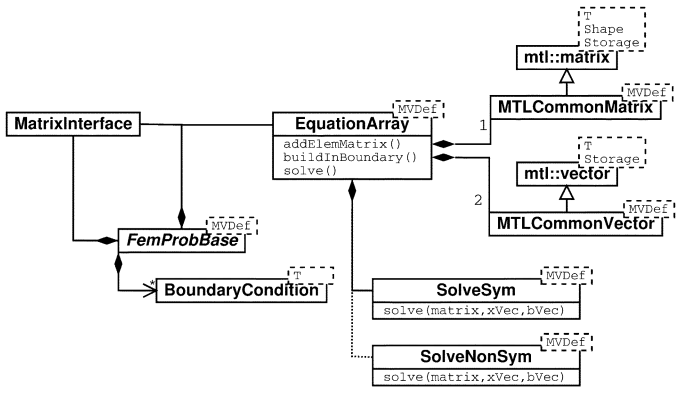

Fig. 4. Class diagram showing the inclusion of the MTL/ITL library in the problem solving classes.

library. This method has typically 50 to 100 lines of code. When defining a formulation with capability for handling nonlinear materials, either a relaxation or a Newton–Raphson algorithm can be added to the newly defined formulation. Again, only the specific matrix building part (e.g., adding the jacobian matrix) has to be coded.

Fig. 4 shows how the representation of matrix and vectors is handled by an external library (MTL [7]). Wrapper classes adapt the interface of to the one needed by the equation array. The solving of the problem is handled by another external library (ITL [8]), which is called from a class linked to the equation array.

III. Available Solvers

In the open-source release, the solvers included are those which were most tested and of general interest. A 2-D solver (iMOOSE.stat2d) for static electromagnetic calculation uses a node-based magnetic vector potential. For 3-D electromagnetic calculation, an edge-based approach with the magnetic vector potential is taken. Additionally, this solver features a pre-processing stage with an electric vector potential for arbitrarily shaped source current regions. Examples of calculations using this solver are depicted in Fig. 5.

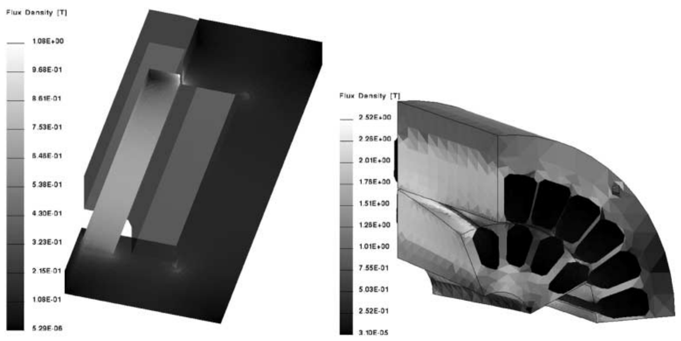
Fig. 5. Team 20 problem and a skewed 3-D induction machine calculated with 3-D static solver iMOOSE.stat3d.

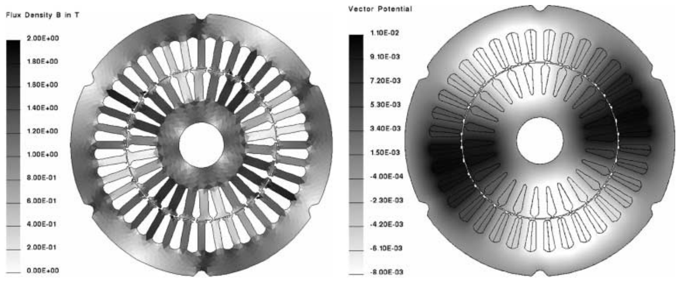
Fig. 6. Flux density and vector potential distribution in an induction machine calculated with iMOOSE.asm2d.

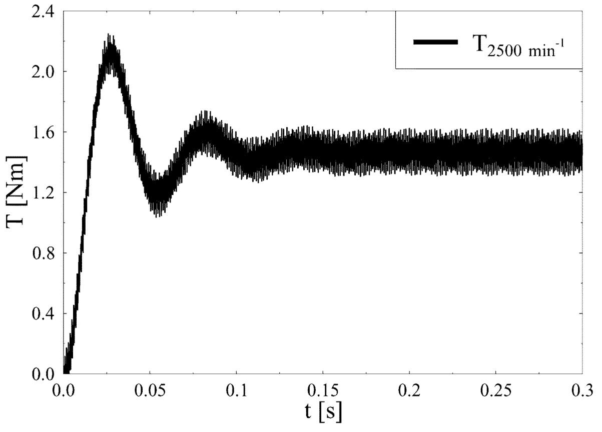

Fig. 7. Time-dependant torque behavior of an induction machine.

A 2-D solver for transient calculation is also included in the open-source release. With this solver, it is possible to take into account eccentric rotor position and external circuits. An example of the use is shown in Figs. 6 and 7.

The most recent addition to iMOOSE is a 3-D solver for time-harmonic calculations using edge elements [9]. It includes three different formulations, all based on the magnetic vector potential $\vec{A}$. A single $\vec{A}$ approach, a combination with an electric vector potential ($\vec{A} - \vec{A}$, $\vec{T}$) and a combination with an electric scalar potential ($\vec{A} - \vec{A}$, V). The three formulations have different applications depending on the nature of the regarded eddy-current regions, the frequencies, and conductivities involved. Fig. 8 shows the results of the TEAM Workshop problem no. 7 [10] used for the verification of the solver.

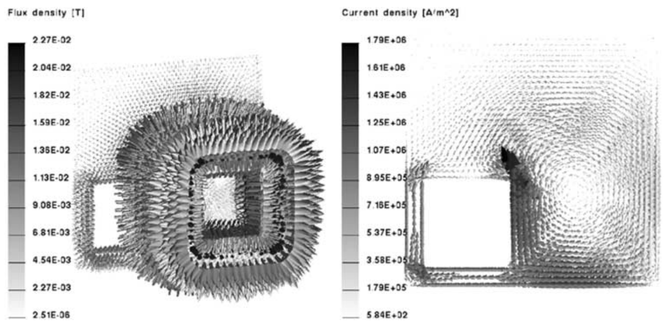
Fig. 8. Flux density and current density distribution for TEAM Workshop problem no. 7.

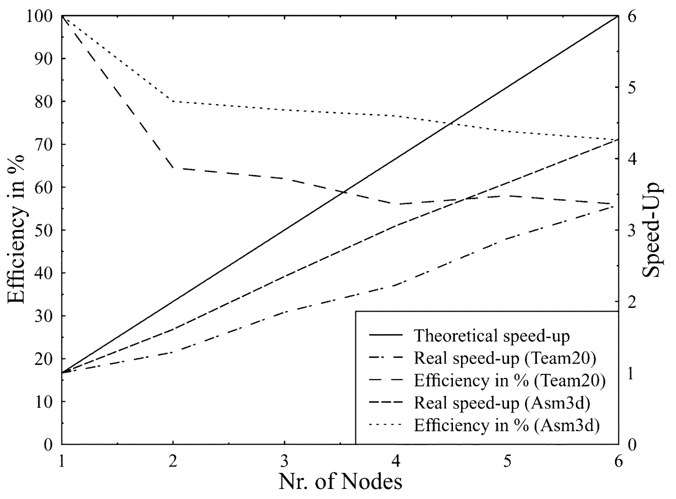

Fig. 9. Efficiency and speedup with parallel execution.

A. Parallel Execution

In order to accelerate computation, the solvers provided by iMOOSE can easily be parallelised. By doing so, cheap clusters of standard personal computers can be used to achieve high speedups. To implement parallel execution, heavy use of open-source libraries and components has been made. Mesh partitioning is done by Metis [11], the equation system is solved by the PetSC-library [12] and the program runs in a MPI environment (LAM [13]).

The above shown examples for 3-D, static calculation (Team 20 Problem—mesh with about 120 000 elements, and an induction machine (ASM3-D)—about 300 000 elements) have been calculated on a cluster of up to six PCs, each with an AMD Athlon 900-MHz processor and 1-GB memory. Fig. 9 shows the achieved speedups and the per-processor efficiency. The theoretical efficiency of 100% is not achieved. For larger problems, the efficiency per processor reaches values between 70% and 80%, e.g., increasing the speed of execution four times when using a six-processor cluster.

IV. Software Engineering

A project the size of iMOOSE is obviously not written by a single programmer. Thus, a team of programmers has to be coordinated. And, as an open-source project, the input of other teams or programmers from other institutions is not only possible but desired. Hence, some care has to be taken with the programming. Coding standards (naming conventions, indentation) help in reading and understanding the source. Documentation of the source code is very important. iMOOSE uses in-line documentation embedded in the source files, from which doxygen [14] generates HTML documentation. A revision control system is used to track changes to the source code, both on the local repository as well as with the open-source published source code, hosted on Sourceforge [15]. Automatic compile and work tests ensure that no errors go unnoticed.

A. Extensibility

One main concern in the design of iMOOSE is to be easily extendible. In a research environment, often new functionalities have to be implemented and tested. The object-oriented design with its defined interfaces, the low coupling between different parts of the class library, and the high cohesion inside the classes makes this possible. The authors themselves have used this extensibility, since iMOOSE has grown with the needs of the developers, implementing different element shapes and formulations, leading to the different solvers presented above. But also external developers could benefit from this possibility, by adding different formulations, e.g., for high-frequency problems or by contributing modern equation-solving algorithms, e.g., multigrid solvers. Also, the interfaces to external meshing or post-processing tools can easily be adapted.

B. Prerequisites

The idea of open-source software makes it possible to profit from other's experience and work, relieving the programmer of the need to reinvent the wheel or implement functionalities not within his core competences. Thus, iMOOSE uses other software projects for some special tasks. The core and the solvers require the presence of the Matrix Template Library [7] (MTL) for the representation of matrices and vectors and the Iterative Template Library [8] (ITL) for solving the equation systems. The parallel execution of solvers using Metis [11] for graph partitioning and mesh distribution, PetSC [12] for parallel solving of equation systems and LAM-MPI [13] for the communication between the nodes. The largest number of external components is required for iMOOSE.trinity, the visualization tool. It uses Python [16] as a scripting language, Qt [17] for the graphical representation and Mesa [18] for 3-D visualization.

Since iMOOSE is intended mainly for development and research and not distributed as a product, only a source-code package is available. Compilation by the user is required. The language used is C++, which is a rather recent development, that became a standard in 1998. Nevertheless, compilers had difficulties in keeping up with the standard, and thus C++ is still only portable with some effort. The development of iMOOSE started in 1997, and has therefor also evolved with the language and the compilers used. The main development takes place in an GNU/Linux environment.

Another prerequisite for the use of the iMOOSE package is an external mesh generator. The authors use Ansys [19], although any program with a documented mesh exchange format could be used. An open-source alternative is still missing.

V. Conclusion

An object-oriented class library and program package for finite-element computations has been presented. It is released as an open-source project available to everyone.

References

[1] G. Arians, T. Bauer, C. Kaehler, W. Mai, C. Monzel, D. van Riesen, and C. Schlensok. Innovative Modern Object-Oriented Solving Environment—iMOOSE. [Online]. Available: http://www.imoose.de.

[2] R. C. Mesquita, E. J. Silva, R. L. Braga, M. A. Matias, C. R. S. Nunes, J. A. Vasconcelos, C Jr, A. M. de Oliveira, J. P. A. Bastos, M. C. Costa, L. Lebenzstajn, A. B. Dietrich, J. R. Cardoso, and S. S. S. Melnikoff, "An object-oriented application framework for the computation of electromagnetic fields," in *Proc. CEFC'02*, Perugia, Italy, 2002, p. 256.

[3] S. Giurgea, T. Chevalier, J. L. Coulomb, and Y. Marechal, "Unified physical properties description in a multi-physics open platform," in *Proc. CEFC'02*, Perugia, Italy, 2002, p. 253.

[4] F. Henrotte, B. Meyes, A. Genon, and W. Legros, "An object-oriented decomposition of the f.e. procedure," *IEEE Trans. Magn.*, vol. 32, pp. 3403–3406, May 1996.

[5] G. Arians, D. van Riesen, and G. Henneberger, "Innovative object oriented environment for designing different finite element solvers with various element types and shapes," in *Proc. 13th Compumag*, 2001, pp. 218–219.

[6] C. Larman, *Applying UML and Patterns*. Englewood Cliffs, NJ: Prentice-Hall, 1998.

[7] A. Lumsdaine, J. Siek, and L.-Q. Lee. The Matrix Template Library—Mtl. [Online]. Available: http://www.osl.iu.edu/research/mtl

[8] ——, The Iterative Template Library—Itl. [Online]. Available: http://www.osl.iu.edu/research/itl

[9] D. van Riesen, C. Kaehler, and G. Henneberger, "Mixing nodal and edge elements in an object-oriented fem calculation tool," in *Proc. EMF 2003*, Aachen, Germany, pp. 301–304.

[10] K. Fujiwara and T. Nakata, "Results for benchmark problem 7 (asymmetrical conductor with a hole)," *Compel*, vol. 9, pp. 137–154, Apr. 1990.

[11] G. K. Karypis *et al.*. Metis—Family of Multilevel Partitioning Algorithms. [Online]. Available: http://www-users.cs.umn.edu/karypis/metis/

[12] Petsc—The Portable, Extensible Toolkit for Scientific Computation. Various. [Online]. Available: http://www-fp.mcs.anl.gov/petsc/

[13] LAM/MPI Parallel Computing. Lam Team. [Online]. Available: http://www.lam-mpi.org

[14] Doxygen. Dimitri. [Online]. Available: http://www.doxygen.org/

[15] Sourceforge.net, the World's Largest Open Source Software Development Website. Various. [Online]. Available: http://www.sourceforge.net/

[16] Python. Python Software Foundation. [Online]. Available: http://www.python.org/

[17] The qt Library. Troll Tech Inc.. [Online]. Available: http://www.trolltech.com/products/qt/index.html

[18] B. Paul. The Mesa 3d Graphics Library. [Online]. Available: http://www.mesa3d.org

[19] Ansys. Ansys Inc.. [Online]. Available: http://www.ansys.com