Object-Oriented Design of Finite Element Calculations with Respect to Coupled Problems

Wolfgang Mai and Gerhard Henneberger

***Abstract*—This paper presents a new object-oriented design of software for finite element calculations. Special attention is given to coupled problems with nonlinear materials. Fundamental ideas of object-oriented design, especially high cohesion, low coupling and encapsulation of classes, are strictly taken into account. The concept is guided by the idea to reuse as many parts as possible. The hierachy of classes for the elements, materials and field problems can be extended easily by specialization.**

***Index Terms*—Finite element methods, object-oriented methods, object-oriented programming.**

I. Introduction

IN RECENT years the three words *object-oriented analysis* (OOA), *object-oriented design* (OOD) and *object-oriented programming* (OOP) have received attention in the design of finite element tools [1]–[7]. The essence of object-oriented analysis and design is to emphasize considering a problem domain and logical solution from the perspective of objects [8]. During object-oriented programming the designed components are implemented into computer language [9].

Because the finite element analysis (FEM) consists of parts which can be treated as objects, e.g. matrixes, vectors, meshes and elements, the object-oriented concepts are good candidates for designing finite element tools.

This paper proposes concepts meant to reduce the time of implementing new problems, different shapes of elements or new formulations of materials, especially for coupled nonlinear problems, using object-oriented techniques. Because of information hiding (encapsulation) the modification of existing code is minimized and the overall class library is more reliable. Special attention is also given to the low coupling and high cohesion principles.

The use of specialization leads to a concept, in which the description of the field problem, the postprocessing, the error calculation and the adaptive remeshing depend neither on the solution order nor on the nonlinearity of the materials. This is realized by Gauss point integration and specializations of the classes `Elem` and `MatProp`. The designed concept is programmed by the authors in C++ [10], [11]. In this paper the proposed concepts are presented along with the class and collaboration diagrams.

Manuscript received October 25, 1999.

The authors are with RWHT Aachen, Institute for Electrical Machines, Schinkelstrasse 4, 52062, Aachen, Germany.

Publisher Item Identifier S 0018-9464(00)04964-5.

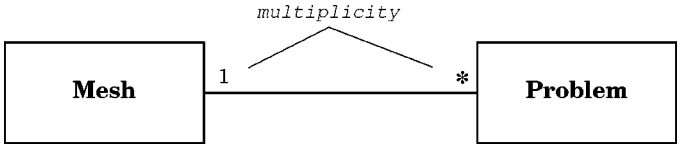

Fig. 1. The association with multiplicity and role.

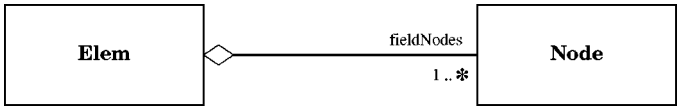

Fig. 2. An element consists of a set of field nodes.

II. Class Diagrams of the Concept

Objects are defined as a concept or abstraction with boundaries and meaning for the problem [12]. A class describes a group of objects with similar properties (attributes), common behavior (methods) and common relationships to other classes (associations).

This paper applies the Unified Modeling Language (UML) as notational system to describe the object-oriented concepts [13]. One view of the object-oriented model is the static view. The main diagram of this view is the class diagram, which displays the classes and the associations among them. In the following subsections different parts of the proposed class diagram are presented.

A. Association Between the Classes `Mesh` and `Problem`

Coupled FEM problems consist of one or more finite element meshes and several field problems, e.g. electromagnetic and thermal problems [14]. In this paper both the mesh and the problem specification, i.e. what kind of problem is considered, are treated as separate objects (high cohesion). Fig. 1 shows the two associated classes `Mesh` and `Problem` in a class diagram using the UML. Each solver is applied on one mesh whereas one mesh can serve many problems. This is displayed by the multiplicity on both sides of the association in Fig. 1.

B. Elems and Nodes

All nodal based finite elements have many things in common. They have a label, an ID and are given by a set of nodes. These informations are saved in the class `Elem` The fact that every element consists of one or more field nodes is the basis for a *has-a* association in the object-oriented model, shown in Fig. 2.

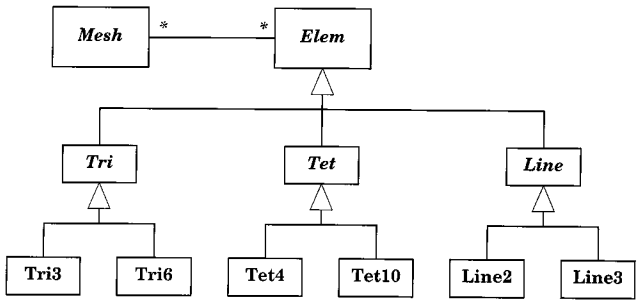

Fig. 3. The generalization of the element classes.

This is called an aggregation and is represented by a hollow diamond.

C. Hierarchy of Elements

All element shapes are derived from the class `Elem` by specialization. Specialization refers to the fact that a subclass refines or specializes a superclass, e.g. `Elem` shown in Fig. 3 with small triangles.

Many methods of `Elem` are abstract, i.e. they are offered but not defined in that class. Such methods are "*the number of sides*" or "*calculate the values of the interpolation functions*." The subclasses, e.g. `Tri3`, inherit all declarations of the abstract methods and define them. `Tri` knows the number of sides, whereas `Tri3` is the expert of the interpolation functions. Methods like "*calculate and return the element matrix* $\int \mathrm{grad}\alpha_i \mathrm{grad}\alpha_j \, d\Omega$" are defined in `Elem` because of the realization by Gauss point integration, which is basically the same for all used elements. The different number and positions of the Gauss points are provided by the shape classes, e.g. `Tet`, via abstract methods, because of the independency of order.

D. Problem Classes and their Associations

Since finite element formulations have relationships to boundary conditions and to a mesh regardless of the type of problem, all problems in the proposed design are subclasses of `Problem`, see Fig. 4. This class also aggregates the class `EqunArray`, which consists itself of the system matrix, the right value vector and the unknown solution vector and is able to solve this linear set of equations. Note that `Problem` takes care of all communications to the mesh and to `EqunArray`, including the mapping of the global node numbers on the matrix rows (encapsulation). The designing engineer doesn't even *see* the system matrix, let alone any handling. He is only concerned with building the element matrix and defining the used material properties.

To add a new problem to the library, only the used formulation has to be designed in a subclass. That is shown in Fig. 4 for two electromagnetic potentials $\vec{T}$ and Ω and the temperature ϑ. The representation of the material properties is discussed in the following subsection, while the Section III-A. explains the construction of the system matrix.

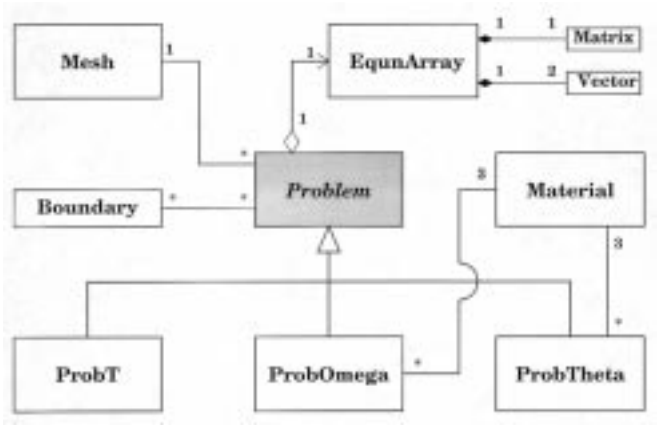

Fig. 4. The class diagram of the problems and their associations.

E. Material Representation

Special attention is given to the design of materials. All problems operate with the super class `MatProp`, which in fact handles one specific material property. In order to identify materials, one can design a class `Material`, which aggregates the used material properties as a set of `MatProp` classes.

The class `MatProp` is considered a superclass. In the case of a simple geometrical distribution of the property, e.g. the property is constant over the mesh or for each label, the class is specialized by the subclasses `MatPropConst` or `MatPropConstLabel` respectively. Fig. 5 shows the designed subclasses of `MatProp`.

In the case of a more advanced material distribution in the problem the classes `MatPropDepLin` and `MatPropDepTable` are added to the class library. Each of them allows the use of a material property, which depends on one solution of another field problem. For example in a coupled electro-thermal problem the electrical conductivity depends on the temperature of the material. The dependence is defined by a linear or approximated curve as shown in Fig. 5. After choosing the appropriate class, the key points of the curve are set and the class is associated to the field solution, on which it depends. After that, the material property is used in the problem definition without considering this dependency anymore, because only the superclass `Matprop` is asked for the values. In fact, the correct property for a given element is calculated automatically and hidden for the problem. Changing from a simple material to a dependent one does not cause any change in the matrix building operation.

Because of the proposed modeling of meshes, problem definitions and material properties, one can easily program a solver for coupled problems. Within a class, each object is independent, because all relevant informations are encapsulated inside it.

III. Dynamic View

The dynamic view describes the dynamic behavior of the object-oriented model. The counterpart of class diagrams are collaboration diagrams. They show objects along with links, i.e. instantiated associations. They also display the messages between

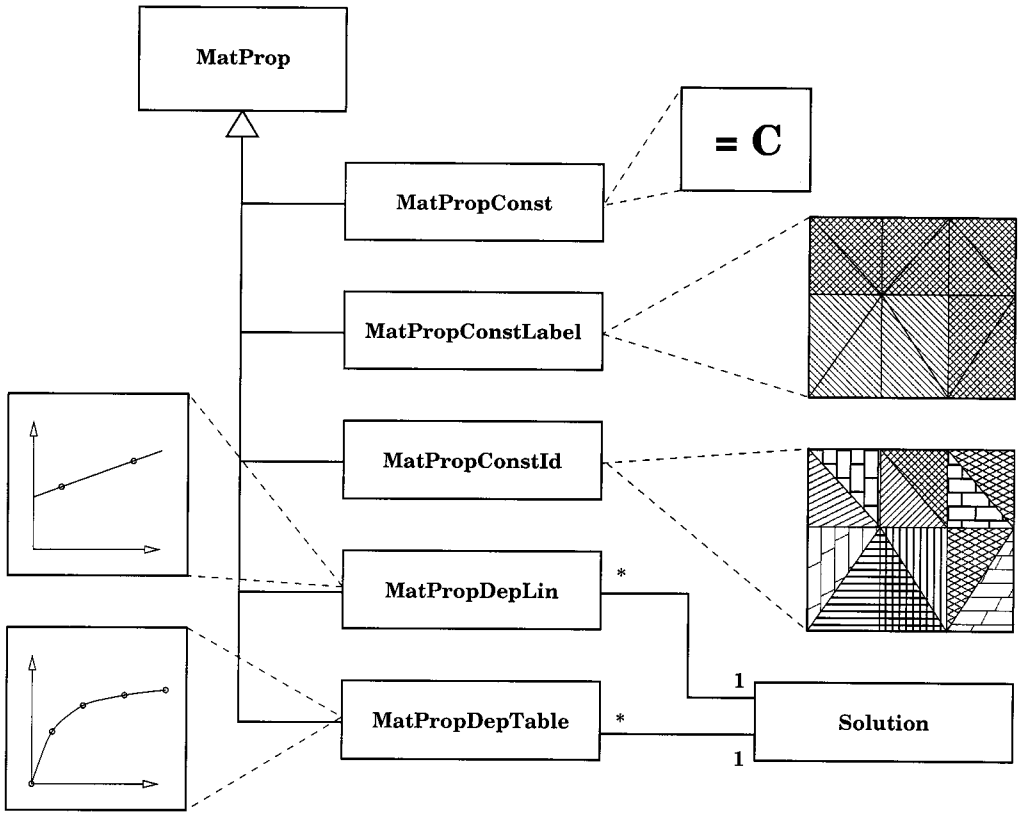

Fig. 5. The material classes and their superclass.

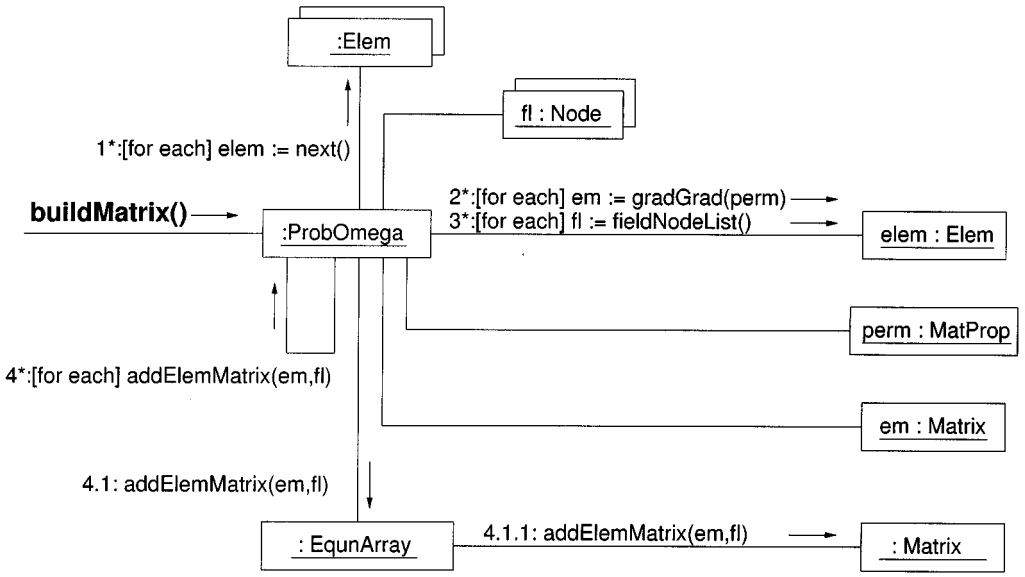

Fig. 6. Building the system matrix.

the objects, each of these messages triggers an operation inside an object.

A. Construction of the System Matrix

This subsection shows that the operation which builds up the system matrix considers neither the element shape nor the order. This is all hidden in the class representing the actual element. Therefore, this operation does not have to be changed when another type of element with a different order is given to the solver. No *if-clauses* are necessary.

Fig. 6 shows the collaboration diagram of this operation. An object of the class `ProbOmega` is asked to build the system matrix. In UML the name of an object is underlined.

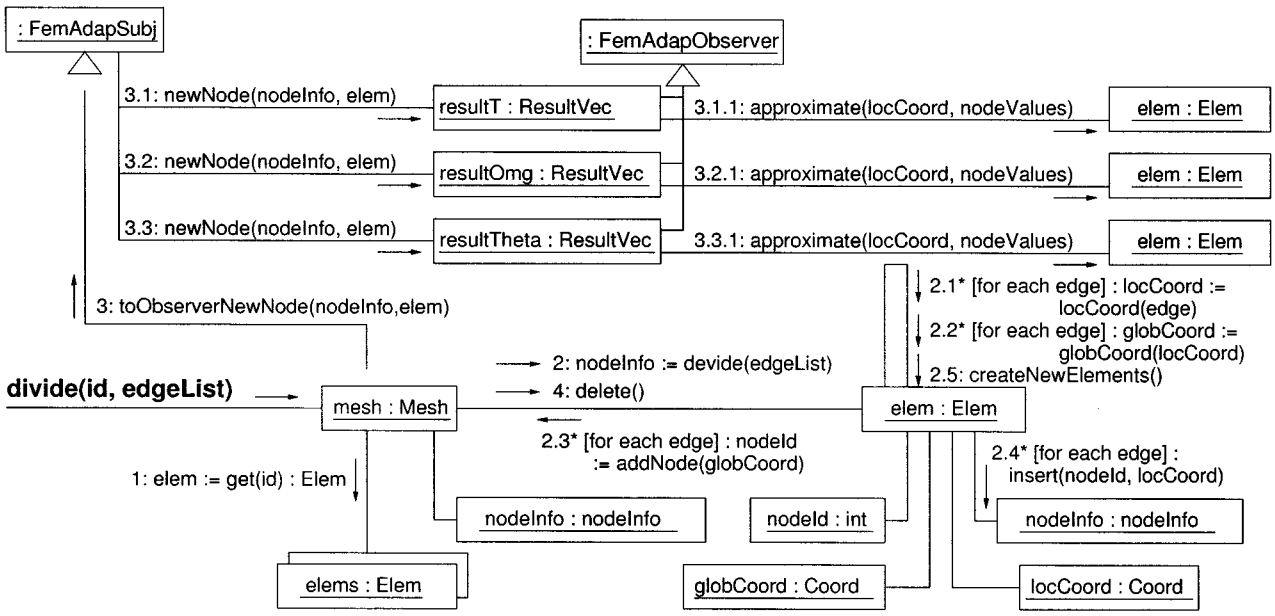

Fig. 7. Automatical approximation of the solution in case of adaptive refinements of a finite element mesh.

This object knows the object `perm`, which stands in this example for the permeability. As well known, the task is a loop over all elements (step 1). Each element, shown as `elem`, is requested to build the element matrix by the use of the material (step 2). It also returns a list of the field nodes (step 3). Then both the matrix (`em`) and the list (`f 1`) are given to the operation add`ElemMatrix` (step 4).

Note that the operation `addElemMatrix` is defined in the superclass `Problem`. Therefore the engineer who adds this new problem class `ProbOmega`, does not see how this operation handles the received element matrix. Each node ID (the ID's can be numbered non consecutively) is mapped on the corresponding number of the matrix's row. Then both the row numbers and the element matrix are sent to the equation array (step 4.1). This adds the element matrix to the system matrix, an operation which is again hidden to the objects outside.

Fig. 6 shows the benefits obtained by taking strictly into account the principles of low coupling and high cohesion. The way an object performs its tasks can be changed without causing changes in other objects. Also, another object can be used instead. For example, in Fig. 6 the object `EqunArray` can be substituted by a faster one recently developed.

B. Mesh Refinement

Another part of the proposed object-oriented model manages the automatic approximation of node solutions in the case of adaptive mesh refinements. Algorithms for the remeshing itself are considered as known and are not explained here.

The basic idea of the proposed model is the following: If any element of any mesh is subdivided, then all solutions based on this mesh should be informed automatically. They should then resize the number of saved values and approximate the new values based on the shape functions of the divided element. This approximation process should also run automatically. This is especially of interest in coupled problems, where a number of solutions rely on a mesh and it is necessary to have up-to-date solutions because of the material dependencies modeled with the class `MatPropDepTable` [14].

The flow of information is explained by the help of Fig. 7. The main objects of this large diagram are `mesh`, `elem`, `FemAdapSubj` and `FemAdapObserver`. The observer pattern is applied [10].

The mesh is asked to divide the elem at given edges. After the element is identified (step 1), it received the request to divide itself (step 2). Each element knows the rules to refine itself It determines for each given edge, both the local and the global coordinates (step 2.1 and 2.2). Now it requests new nodes from the mesh, because creating nodes is out of the element's scope. This step 2.3 returns the new node ids. For each edge both the node id and the local coordinates are saved in the object `nodeInfo` (step 2.4). Finally, step 2.5 creates new elements and the operation (step 2) is fulfilled. The control is returned to the mesh.

In this moment the new nodes are known and the old element is still part of the mesh, this is the only chance to approximate the solution values for these nodes based on the old element. All objects, which are interested in remeshing information, i.e. the shown objects `ResultVec`, are subclasses of `FemAdapObserver` and had registered at the mesh, which is a subclass of `FemAdapSubj`.

Each solution object is sent the old element and the object `nodeInfo` info (step 3). The solution-objects can now easily resize the vector and ask the element for the values at the positions of the new nodes. This is done inside the element by approximation of the received `nodeValues`, i.e. the values of the field nodes. Now the old element is deleted (step 4).

Note that, while the figure looks difficult, the implemented source code consists of only a few lines, in fact in rough approximation one for each numbered step.

IV. Conclusions

This paper presents a new object-oriented design for a finite element tool. It proposes the realization of elements, field problems and materials. Strong consideration of the principles high cohesion, low coupling and encapsulation results in a set of classes, which can easily be extended.

Also, different types of field problems can be combined. These coupled problems can be defined on different or identical meshes. The entire problem definition is handled as one single object.

The presented dynamic view shows the construction of the system matrix and the automatical approximation of field solutions in the case of adaptive mesh refinements. Every solution of a coupled problem has always the correct number of values (corresponding to the number of nodes).

Most parts of the design and of the source code do not depend on the element shape or order. The result is heavy reuse of existing code. The code can be easily read and maintained.

References

[1] R. C. Mesquita, R. P. Souza, T. Pinheiro, and A. L. C. C. Magalhaes, "An object-oriented platform for teaching finite element pre-processor programming and design techniques," *IEEE Trans. Magn.*, vol. 34, no. 5, pp. 3007–3410, September 1998.

[2] M. Popescu, I. Munteanu, C. G. Constantin, and D. Ioan, "An object oriented data structure for field analysis," in *Proceedings of the Eight Biennial IEEE Conference on Electromagnetic Field Computation (CEFC)*, Tucson, USA, 1998.

[3] E. J. Silva and R. C. Mesquita, "Data management in finite element analysis programs using object-oriented techniques," *IEEE Trans. Magn.*, vol. 32, no. 3, pp. 1445–1448, May 1996.

[4] F. F. N. Rocha and R. C. Mesquita, "An object-oriented data structure for a 3-D electromagnetic field computation program pre-processor," *IEEE Trans. Magn.*, vol. 32, no. 3, pp. 1449–1452, May 1996.

[5] E. J. Silva, R. C. Mesquita, R. R. Saldanha, and P. F. M. Palmeira, "An object-oriented finite-element program for electromagnetic field computation," *IEEE Trans. Magn.*, vol. 30, no. 5, pp. 3618–3621, September 1994.

[6] F. Henrotte, B. Meys, A. Genon, and W. Legros, "An object-oriented decomposition of the F.E. procedure," *IEEE Trans. Magn.*, vol. 30, no. 5, pp. 3618–3621, September 1994.

[7] T. Chelcea and D. Ioan, "A hierarchy of classes for the finite element method," in *Proceedings of the Seventh Biennial IEEE Conference on Electromagnetic Field Computation (CEFC)*, Okayama, Japan, 1996.

[8] C. Larman, *Applying UML and Patterns*: Prentice-Hall, Inc., 1998.

[9] B. Stroustrup, *The C++ Programming Language*, 3rd ed: Addison-Wesley GmbH, 1998.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*: Addison-Wesley, 1995.

[11] U. Breymann, *The C++ Standard Template Library*. Munich/Wien: Addison-Wesley, 1996.

[12] J. Rumbaugh, *Object-Oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice-Hall International, Inc., 1991.

[13] G. Booch, *The UML Specification Documents*. Santa Clara, CA: Rational Software Corp., 1997.

[14] W. Mai and G. Henneberger, "Calculation of the transient temperature distribution in a TFIH device using the impedance boundary condition," *IEEE Trans. Magn.*, vol. 34, no. 5, pp. 3094–3097, September 1998.