OOFEM – An Object Oriented Framework for Finite Element Analysis

B. Patzák, Z. Bittnar

This paper presents the design principles and structure of the object-oriented finite element software OOFEM, which has been under active development for several years. The main advantages of the presented framework include modular design, extensibility, and robustness. The code itself is freely available and is distributed under GNU public license. It provides tools for linear and nonlinear analysis of mechanical and transport problems on sequential and parallel computers.

Keywords: finite element software; object oriented FEM design.

1 Introduction

In recent years, research community has faced growing demands to merge the power of current computer technology with the extensive knowledge acquired in engineering, mathematics, physics and other disciplines, and to exploit them effectively in the design of new multi-functional materials and structures.

Instead of sticking to traditional (usually very conservative) design code formulas and provisions, modern design procedures should focus on optimizing the actual performance, taking into account multiple objectives and criteria, such as sufficiently low probability of failure, low cost, long-term durability, proper functionality and high utility, compatibility with the environment, aesthetic quality, etc. This is not possible without powerful modeling, simulation and optimization tools supporting the designer in the decision-making process. The behavior of structures, solids, and fluids is governed by complicated systems of partial differential equations with appropriate boundary conditions. A reliable and accurate analysis of this behavior, necessary for practical decisions regarding design, optimization, risk assessment etc., is frequently based on a numerical simulation and requires the development of efficient computational tools.

These growing demands for realistic modeling that typically includes state-of-the-art constitutive models, adaptive and multi-level solution techniques brings in new software issues. A very important feature of any modern computational code is its open nature, so that it should allow straightforward and efficient implementation of new solution methods, algorithms, material models, etc. An analyst or researcher naturally wants to work with a code which is easily extensible towards future demands, easily maintainable, but still efficient and portable across many platforms. Object-oriented modelling is a tool that has been successively used to design and implement complex software systems meeting the above criteria. It is based on the uniform application of the principles for managing complexity – abstraction, inheritance, association, and communication using messages. The design of an object-oriented application consists in finding classes and objects, identifying structures and attributes, and defining the required services.

In recent years, a number of articles on applying an object-oriented approach to finite element analysis have been published. In 1990 Fenves [1] described the advantages of an object-oriented approach for developing of engineering software. Forde et al. [2] presented one of the first applications of object-oriented programming to finite elements. Many authors have presented complete architectures of OO finite element codes, notably, a coordinate free approach by Miller [3], a non-anticipation principle by Zimmermann et al. [4], Dubois-Pelerin et al. [5, 6, 7], and Commend [8]. Recent contributions include the work of Mackie [9, 10, 11], Archer et al. [12, 13], and Menetrey et al.[14].

This paper presents the design principles and structure of object-oriented finite element code OOFEM [15]. This code has been actively developed for several years and it is distributed as a free software under GNU public licence. The basic intentions of OOFEM design include modularity, open nature, extensibility, maintainability, portability, and last but not least, computational performance. Although the primary focus has been given to research applications, the code has been used several times for solving of industrial problems. In the next section, the general structure of the code is presented using the Coad-Yourdon methodology [16]. Such a representation allows to show class hierarchy as well as the mutual relations between the classes, representing the generalization/specialization, whole/part, or association relations. All the fundamental abstract base classes, representing the basic building blocks of finite element code, will be introduced and their role will be discussed. Finally, the OOFEM features and future development directions will be presented.

2 Design principles

The overall structure consists of several modules. The core module is called OOFEMlib. It contains the definition of fundamental top-level FE classes, that represent, for example, degrees of freedom, nodes, elements, integration points, boundary and initial conditions, constitutive models, numerical solvers, sparse matrices, and problems under consideration. It also contains some utility classes that are of general use and that can facilitate development, like representations of vectors and matrices, etc. This module introduces the fundamental class hierarchy, which is intended to be general enough to incorporate any FE problem and which is, at the same time, problem independent. The primary role of these core classes is to specify a general interface that defines the services that are provided by each derived class. These services are typically abstract ones, they are implemented by

inherited classes, which implement particular objects. The role of abstract services is very important, since they declare the general interface, which is implemented by derived classes. Thus any derived class is enforced to implement this interface, which allows a high level of abstraction.

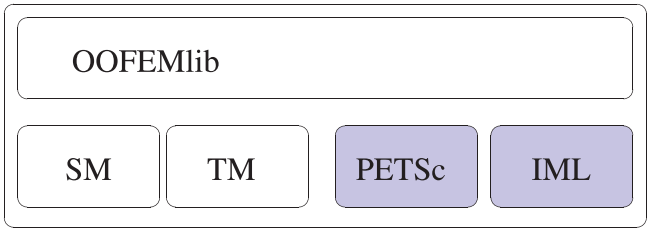

Fig. 1: OOFEM modules

An important consequence of the abstract interface concept is that it allows to implement some general services already at the abstract level. A typical example is stiffness matrix computation, which can be done already at the abstract level, provided that methods for computing the geometrical matrix and material stiffness are declared in an interface specification – they are only declared as abstract (virtual), and implementation is left to derived classes representing particular finite elements. Typical implementation of this procedure then consists in a loop over finite element integration points, and computation of the products of these matrices and summation of the contributions. Implementation of such general services can significantly facilitate the development of new elements. At the same time, such a default service can be overloaded (specialized) by a particular element implementation to reflect the specific needs of particular element formulation, if necessary.

Abstract interfaces allow to a developer to implement high-level functionality using the general interface, without regarding the details of each derived class. And on the other hand, it allows to implement a particular class without deep knowledge of the whole code structure; it is only necessary to implement the required services that constitute the general interface. Such an approach allows to write high level procedures that will work even with classes added in the future. Moreover, such a concept leads to a maintainable and extensible code structure which enables efficient team-work support. However, it is necessary to carefully design the abstract interfaces declared by top-level classes to be general enough to incorporate future demands.

On the top of the core OOFEMlib module, specialized modules are built (see Fig. 1 and 2). These modules contain application-specific classes that implement the required functionality. They typically contain implementation classes representing problem – specific finite elements, constitutive models, boundary conditions, and solution algorithms. A typical example is a structural analysis module (SM) or a transport-problem module (TM). Modules may also represent an interface to external libraries. Such a module then provides "shell" classes that implement the required interface and translate the messages to external library procedures. The PETSc module providing an interface to Portable, Extensible Toolkit for Scientific Computation (PETSC) [17] is a typical example.

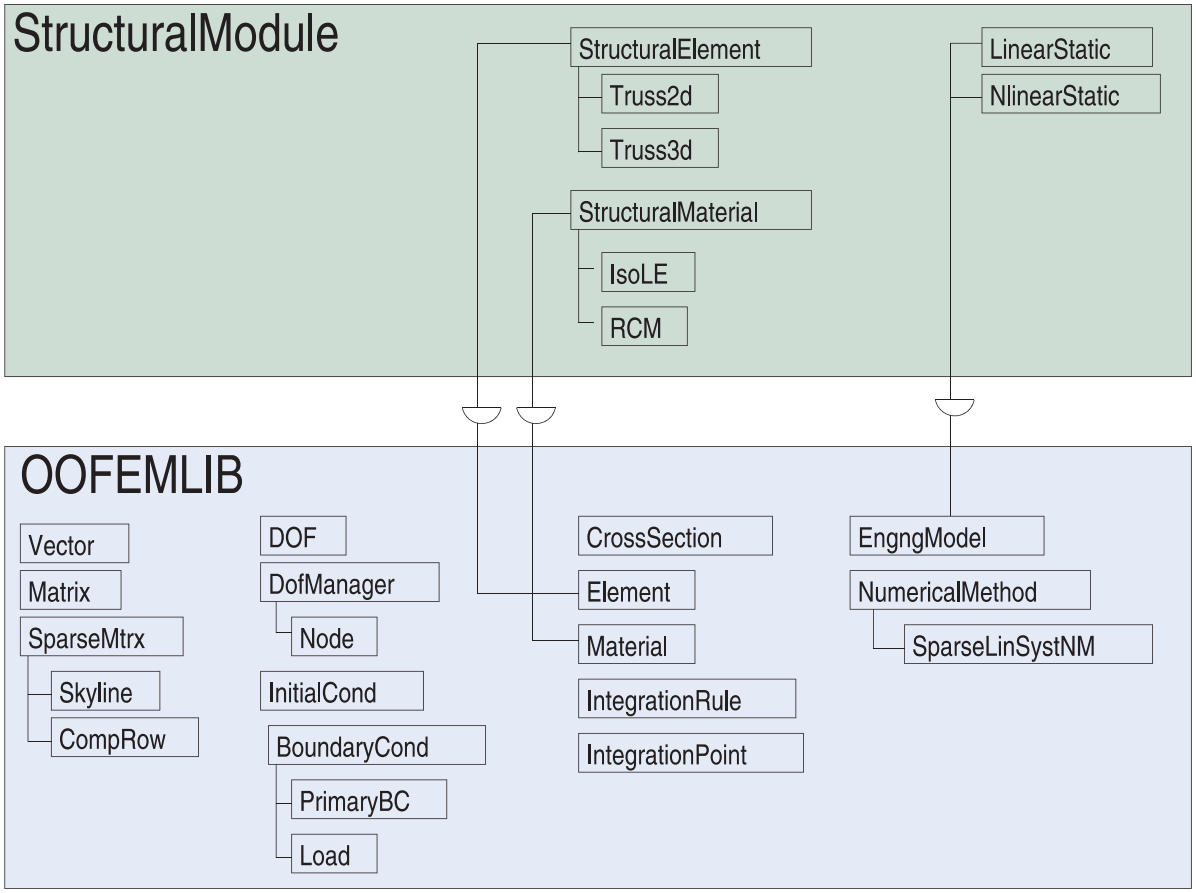

Fig. 2: OOFEM modules – problem independent core module OOFEMlib and problem specific SM module

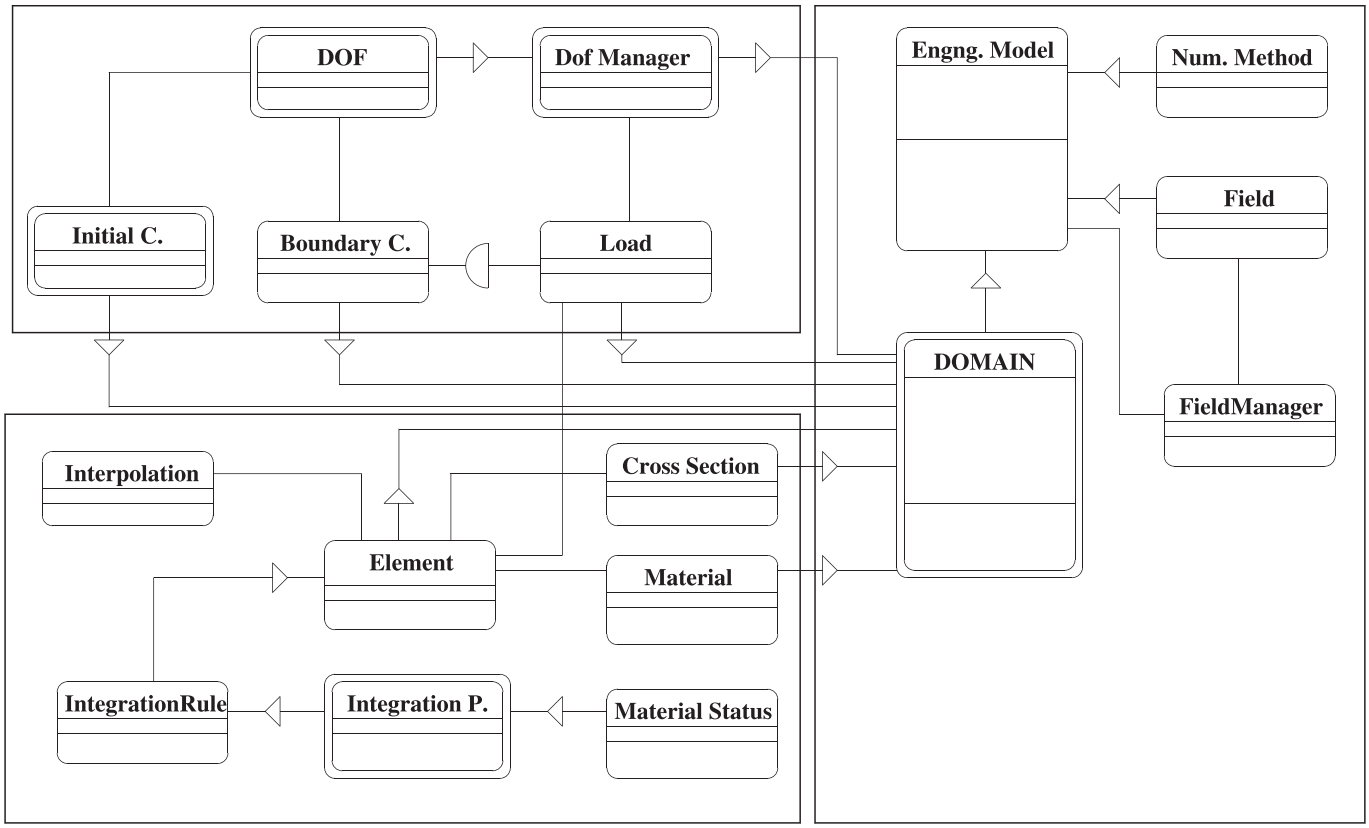

Fig. 3: General structure of OOFEM

3 General structure

The general structure of the OOFEM is shown in Fig. 3, using the Coad-Yourdon representation. In short, abstract classes are represented by single framed rectangles, classes which have instances (so called class&objects) are represented by double framed rectangles. The lines with a semi-circle mark represent a generalization/specialization relation (inheritance), where the line from the semi-circle midpoint points to the parent class. The lines with a triangle mark represent a whole/part relation, where the line starting from the triangle vertex points to "whole" class possessing the "part" class. An association is represented by a solid line drawn between the classes. Bold lines represent communication using messages. The details can be found in [3].

Class DOF represents a single degree of freedom (DOF). It maintains its physical meaning, an associated equation number, and keeps a reference to the applied boundary and initial conditions. The base class DOF manager represents an abstraction for an entity possessing some DOFs. It manages its DOF collection, a list of applied loadings and optionally its local coordinate system. General services include methods for gathering localization numbers from maintained DOFs, computing the applied load vector, and computing transformation to its local coordinate system. Derived classes typically represent a finite element node or an element side, possessing some DOFs. Boundary and initial conditions are represented by corresponding classes. Derived classes from the base BoundaryCondition class, representing particular boundary conditions, can be applied to DOFS (primary BC), DOF managers (typically nodal load), or elements (surface loads, Neumann, or Newton boundary conditions, etc.)

3.1 Problem representation

The problem under consideration is represented by a class derived from the EngngModel class. Its role is to assemble the governing equation and use a suitable numerical method (represented by a class derived from the NumericalMethod class), to solve the system of equations. The discretization of a problem domain is represented by the class Domain, which maintains lists of objects representing nodes, elements, material models, boundary conditions, etc. The Domain class is an attribute of the EngngModel class and, in general, it provides services for accessing particular components. For each solution step, the EngngModel instance assembles the governing equations by summing up the contributions from the domain components. Since the governing equations are typically represented numerically in a matrix form, implementation is based on vector and sparse matrix representations to efficiently store the components of these equations. Then a suitable numerical method, represented by an instance of the class derived from the NumericalMethod class, is used to solve the problem. An important consequence of abstract interfaces is that problem formulation can use any sparse matrix representation and any suitable numerical method, even added in the future, because they all implement the same common interface.

An abstraction for the general field is provided. Fields have the capability to represent any global field like displace-

ment or temperature fields, described using nodal values, and to evaluate the field values at any valid point of the problem domain. A particular problem implementation can store its solution in the form of field(s). This can help significantly, when implementing adaptive or staggered solution techniques, since transfers of solution fields between several grids are provided by the field implementation.

High level numerical methods are represented as a hierarchy of classes derived from the base NumericalMethod class. Classes directly derived from this base class define the problem-specific interface for particular numerical problems (for example, interfaces specific to an eigen value problem or a linear system of equations). The derived classes then implement particular algorithms. The methods forming problem-specific interfaces accept parameters in the form of abstract classes representing sparse matrices or vectors. Thus there are no assumptions about a particular type of data representation. As a consequence, numerical method implementation can work in principle with any sparse matrix, provided that it uses only operations available in the general interface of the basic SparseMatrix class. This is illustrated in Fig. 4, where linear static analysis can use different solution algorithms for a linear system of equations, since they all implement the same interface (here represented by the method "solve"). At the same time the iterative solver can work with different sparse matrix representations, since in principle the only required operation is the multiplication of the atrix by a vector, which is a part of the general sparse matrix interface (in reality, suitable preconditioning should also be applied, but this is omitted here, for the sake of brevity).

The independent problem formulation and the numerical solution, together with independent data storage representation on a numerical algorithm, are the key features that characterize the design and structure of this frame.

3.2 Material-element frame

In this section, the structure of the material-element frame will be described. The primary goal during the design was to achieve straightforward extensibility and a high level of modularity. To achieve these requirements in the context of this frame, the following set of fundamental abstract classes is introduced to represent finite elements (Element class), cross section models (CrossSection class), constitutive models (Material class), integration rules (IntegrationRule class), integration points (IntegrationPoint class), interpolation functions (Interpolation class), and material mode specific containers for storing history variables in integration points (MaterialStatus class).

To reflect the needs of specific problems under consideration, specialized abstract interfaces for particular problems are needed. One should have, for example, a different material model interface for structural mechanics problems and for heat and mass transport problems. These problem – specific interfaces are declared by corresponding problem--specific classes, derived from base classes representing a finite element, a cross section, or a material model. Specific finite elements, cross section models, and constitutive models are then derived from these problem-specific base classes in a frame of the corresponding module.

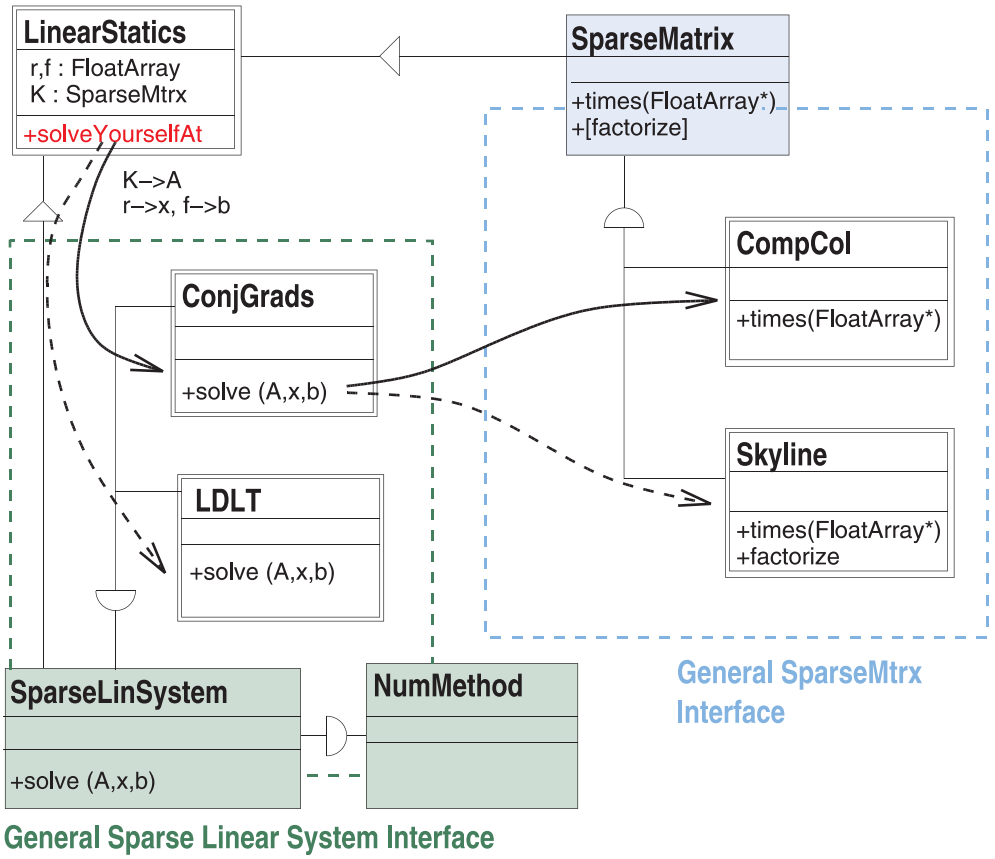

Fig. 4: Independent problem formulation on data storage and numerical algorithm

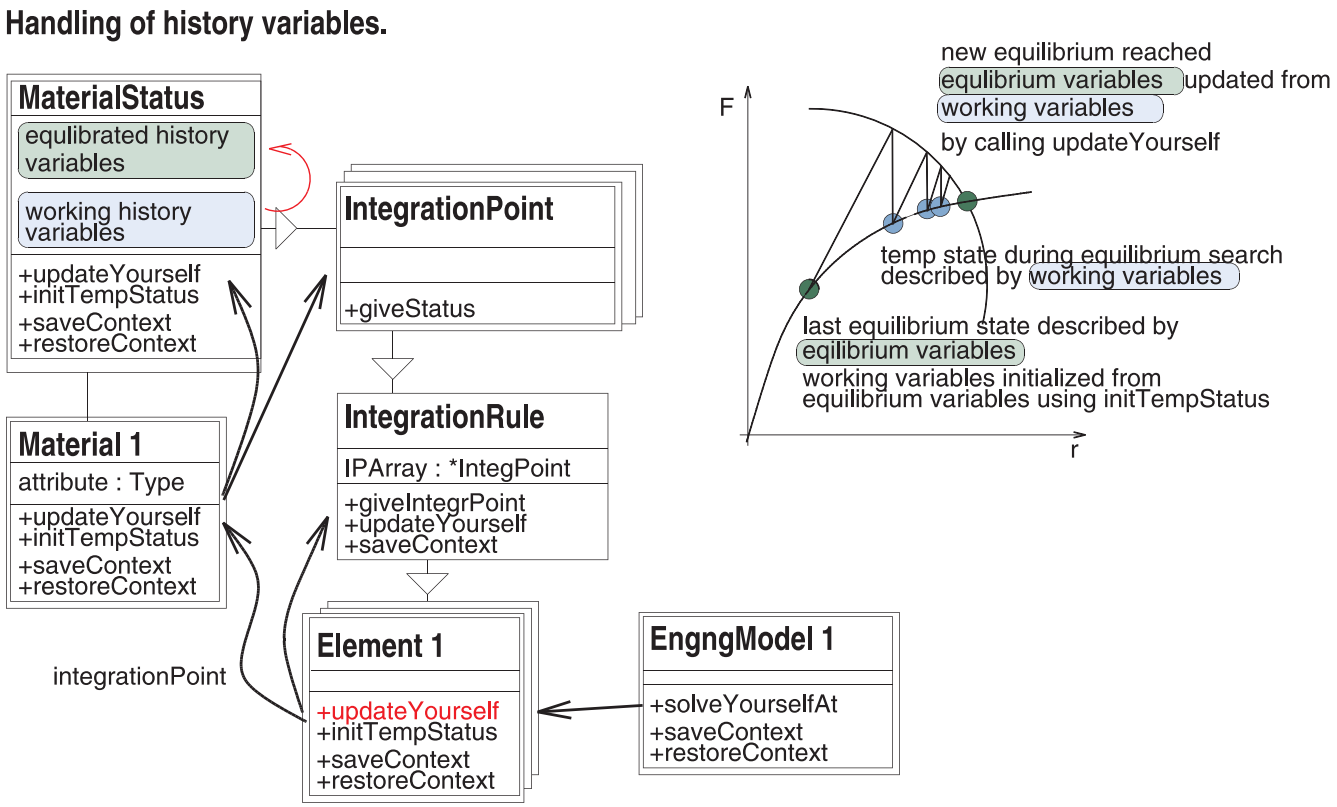

Fig. 5: Constitutive model and its history variables

Particular finite element implementations are represented by the classes derived from the corresponding problem – specific base class. Each finite element can have one or more integration rules, which are abstractions for a set of integration points used to evaluate numerical integrals over an element volume. An integration point maintains its local coordinates and integration weights. Each integration point can maintain one or more instances of MaterialStatus class (the purpose of this feature will be explained later). For convenience, a hierarchy of classes derived from the base Interpolation class, representing FE interpolation, is provided. Derived classes implement many interpolation schemes and can be used to evaluate shape functions and their respective derivatives.

The CrossSection class represents a geometrical model of a cross section. Classes representing finite elements do not communicate directly with a constitutive model. Instead, they always use the CrossSection class interface, which performs necessary integration over the cross section and invokes the corresponding material model services. A cross section model can introduce special integration points to account for a layered description, for example. In such a case, these additional integration points (slaves) are created and stored at every element integration point, but are hidden to element formulation.

A material class represents a base abstract class for all constitutive models. An associated MaterialStaus class is introduced in order to account for extensibility and efficiency requirements. In general, every material model must store its unique history parameters at every integration point. The amount, type, and meaning of these history variables vary from one material model to another. Therefore, it is not possible to efficiently match all needs and reflect them in the integration point data structure. The suggested remedy uses the associated Material Status class, related to the corresponding material model, in order to store the necessary history variables. A unique copy of the corresponding material model status is created and associated to every integration point by the particular constitutive model. The developer of a new constitutive model defines and implements the material class representing the model and it has to define also the associated material status class (derived from base MaterialStatus), which contains the history variables related to the model and corresponding services. Because the integration point is the compulsory parameter of all messages sent to the material model, it can in turn access its related material status from the given integration point, and therefore has access to the corresponding history variables. There are typically two sets of history variables, one related to the previous equilibrium state (needed to correctly evaluate the evolving character of constitutive relations) and the working set, which is changing during the equilibrium iterations (see Fig.5). Once equilibrium is reached, the working set is copied into the equilibrium set. On the other hand, when equilibrium is not reached, the solution step can be restarted, and in this case the working set is initialized from the set related to the previous equilibrium.

Recalling the concept of abstract interfaces, introduction of independent representations for a finite element, a cross section description, and a constitutive model allows to com-

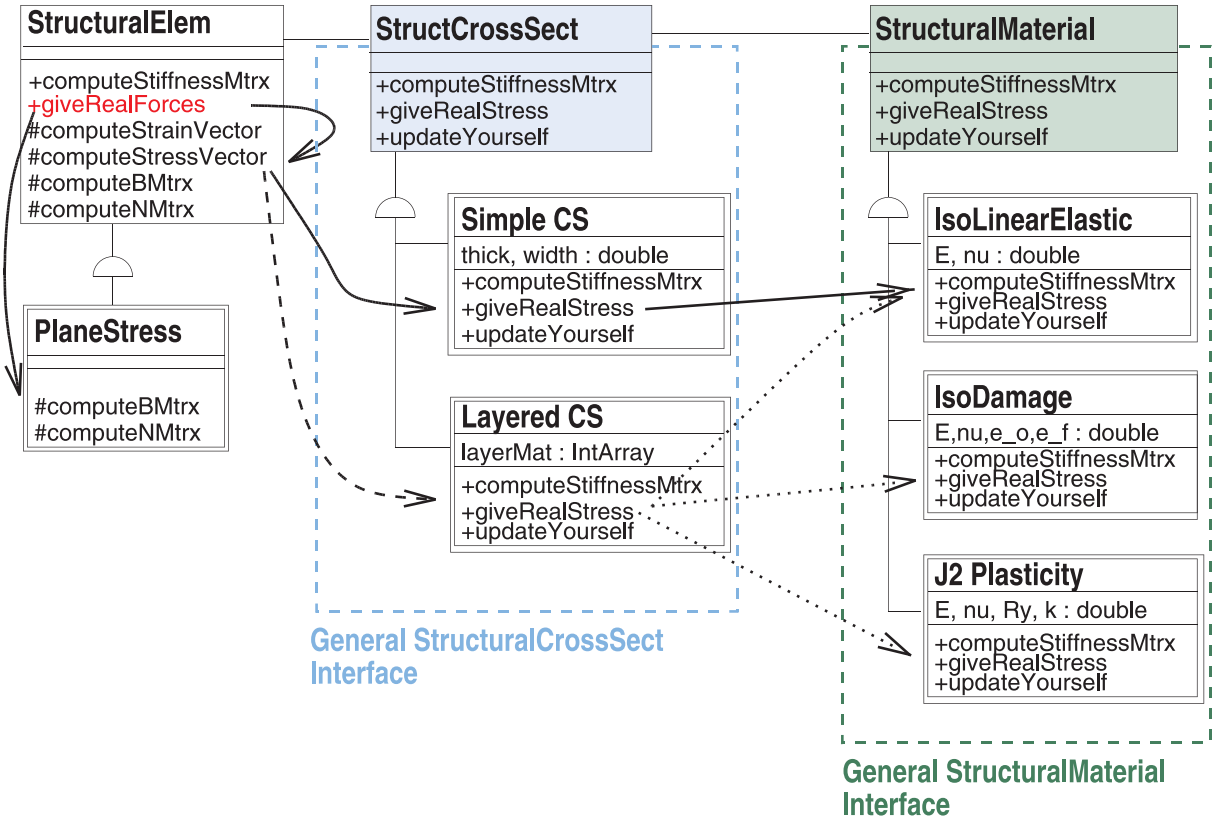

Fig. 6: Cross section and material models interface concept

bine particular a finite element representation with different cross section and material models, see Fig. 6. This is fully transparent, since all cross section and material models implement the same interface, declared by the corresponding abstract classes.

4 OOFEM features

OOFEM is an open source, free software finite element system with object oriented architecture. It is distributed under GNU public license. It is written in the C++ programming language. It operates on various platforms, including Unix (Linux) and Microsoft Windows platforms. A graphical post-processor is available in X-Windows (UNIX).

The general features include staggered solution procedures, a multiple domain concept, full restart support from any saved state, and built-in support for parallel processing (message passing). Many sparse matrix storage schemes are available, as well as the corresponding iterative and direct solvers.

The structural analysis module (SM) includes many analysis procedures including serial and parallel nonlinear static analyses with direct and indirect control, parallel nonlinear explicit dynamics, linear dynamics (eigen value analysis, implicit and explicit integration methods). A large material library including state-of-the-art models for the nonlinear fracture mechanics of quasi-brittle materials and a rich element library are provided.

The transport problem module (TM) is capable of solving a stationary and transient (linear and nonlinear) heat transfer and coupled heat & mass transfer problems. The element library includes axisymmetric, two and three dimensional elements. Staggered analysis of heat transfer analysis and mechanical analysis can be performed, where the temperature field generated by heat transfer analysis can be used in mechanical analysis as temperature loading.

OOFEM interfaces to the following external software: IML++ (template library for numerical iterative methods [18]), PETSC – Portable, Extensible Toolkit for Scientific Computation [17], and VTK (The Visualisation Toolkit [19]).

5 Conclusion

To summarize, a general object oriented environment for finite element computations has been developed. The described structure leads to a modular and extensible code design. Special attention has been focused on important aspects of material-element and analysis frame design. A successful implementation using C++ language verifies the designed program structure and provides a robust computational tool for finite element modeling.

6 Acknowledgment

This work was supported by the Grant Agency of the Czech Republic, under Project No.: 103/04/1394.

References

[1] Fenves G. L.: "Object-oriented programming for engineering software development." *Engineering with Computers*, Vol. **6** (1990), p. 1–15.

[2] Forde B. W. R, Foschi R. O., Stiemer S. F.: "Object-oriented finite element analysis." *Computer and Structures*, Vol. **6** (1990), p. 1–15.

[3] Miller G. R.: "An object-oriented approach to structural analysis and design." *Computers and Structures*, Vol. **40** (1991), p. 75–82.

[4] Zimmermann T., Dubois-Pelerin Y., Bomme P.: "Object-oriented finite element programming Part I. Governing principles." *Comp. Meth. in Appl. Mech. Engng.*, Vol. **98** (1992), No. 3, p. 291–303.

[5] Dubois-Pelerin Y., Zimmermann T., Bomme P.: "Object-oriented finite element programming Part II: A prototype program in Smalltalk." *Comp. Meth. in Appl. Mech. Engng.*, Vol. **98** (1992), No. 3, p. 261–397.

[6] Dubois-Pelerin Y., Zimmermann T.: "Object-oriented finite element programming Part III: An Efficient Implementation in C++." *Comp. Meth. in Appl. Mech. Engng.*, Vol. **108** (1993), p. 165–183.

[7] Dubois-Pelerin Y., Pegon P.: "Object-oriented programming in nonlinear finite element analysis." *Computers and Structures*, Vol. **67** (1998), p. 225–241.

[8] Commend S., Zimmermann T.: "Object-oriented nonlinear finite element programming: a primer." *Advances in Engineering Software*, Vol. **32** (2001), p. 611–628.

[9] Mackie R. I.: "Object-oriented programming of the finite element method." *International Journal For Numerical Methods In Engineering*, Vol. **35** (1992), p. 425–436.

[10] Mackie R. I.: "Using objects to handle calculation control in finite element modelling." *Computers & Structures*, Vol. **80** (2002), p. 2001–2009.

[11] Mackie R. I.: "Object oriented methods and fnite element analysis." Saxe-Coburg Publications, Stirling, UK, 2001.

[12] Archer G. C.: "Object-oriented Finite Element analysis." PhD thesis, University of California at Berkeley, Apr. 1996.

[13] Archer G. C., Fenves G., Thewalt C.: "A new object-oriented finite element analysis program architecture." *Computers and Structures*, Vol. **70** (1999), p. 63–75.

[14] Menetrey P., Zimmermann T.: "Object-oriented non-linear finite element analysis – application to J2 plasticity." *Computers and Structures*, Vol. **49** (1993), p. 767–777.

[15] Patzák B.: OOFEM home page, http://www.oofem.org, 2004.

[16] Coad P., Yourdon E.: "Object-Oriented Analysis." Prentice-Hall, 1991.

[17] Balay S., Buschelman K., Gropp W. D., Kaushik D., Knepley M., McInnes L. C., Smith B. F., Zhang, H.: PETSc home page, http://www.mcs.anl.gov/petsc, 2001.

[18] Iterative Methods Library, http://math.nist.gov/iml++/.

[19] Schroeder W., Martin K., Lorensen B.: "The Visualization Toolkit An Object-Oriented Approach To 3D Graphics." 3rd Edition, Kitware, Inc. publishers, 2003.

Doc. Dr. Ing. Bořek Patzák
phone: +420 224 354 369
e-mail: borek.patzak@fsv.cvut.cz

Prof. Ing. Zdeněk Bittnar, DrSc.
phone: +420 224 354 493
e-mail: bittnar@fsv.cvut.cz

Czech Technical University in Prague
Faculty of Civil Engineering
Thákurova 7
166 29 Prague, Czech Republic

