Advances in Engineering Software 72 (2014) 193–202

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

On design of element evaluators in OOFEM

M. Horák, B. Patzák *, M. Jirásek

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

ARTICLE INFO

Article history:
Available online 12 February 2014

Keywords:
Multi-physics simulations
FEM software design
Object-oriented design
Strong coupling
Damage-plastic model
Coupled heat and mass transfer

ABSTRACT

This paper presents the advanced object-oriented design of finite element representations in a complex multi-physics finite element environment OOFEM [1–3]. The focus is on reuse of existing single-physics capabilities when implementing elements for coupled simulations. This has been achieved by decoupling the description of element geometry, element problem specific capabilities, element interpolation, and integration schemes. The individual problem specific capabilities, represented by a hierarchy of classes derived form *ElementEvaluator*, can be assembled together to define an evaluator for coupled analysis. The presented design leads to an extremely flexible implementation, with clean modular design. The application is demonstrated on an implicit gradient formulation of a damage-plastic model and on coupled heat and mass transfer.

1. Introduction

Numerical simulations are routinely used in research and industry and are accepted as reliable analysis tools. However, in recent years it has become clear that further progress in many scientific and engineering disciplines requires understanding of various complex multi-physics phenomena taking place at different scales of resolution. Therefore, one of the actual challenges in software engineering is to design efficient and modular modeling tools. The aim of this contribution is to present an advanced object-oriented design of a general multi-physics finite element kernel allowing to reuse single-physics formulations in development of coupled multi-physics problems. The available strategies for handling a coupled problem composed of individual components (applications) can be divided into the so-called strongly or weakly coupled schemes, depending on whether consistency of internal values across applications is required after each global time step. In the case of a strongly coupled scheme, the consistency based on global convergence is required, while in the weakly coupled scheme it is not. Weakly coupled schemes are typically solved in an iterative, staggered approach, based on solution fields exchange between individual sub-problems. The strong coupling allows to ensure second-order accuracy and contributes to the increased stability of the coupled algorithm, however its implementation is more demanding. The coupling terms need to be evaluated and a coupled system of equations has to be solved in a monolithic fashion. The focus of this paper is on the development of strongly coupled schemes.

The conventional designs of object-oriented finite element codes introduce an abstraction for finite element, which maintains the description of element geometry, properties and integration scheme and provides services for evaluating characteristic terms, such as the stiffness matrix or the element load vector. In more elaborated designs, a hierarchy of classes is developed, where the base parent element class contains only the problem-independent description (such as element geometry) while specific functionality is implemented by derived classes, representing problem-related base classes. This scheme works well when elements are to be solely used for a specific type of analysis, e.g. for structural analysis.

A problem may arise when one wants to combine and reuse the capabilities of two or more elements to obtain an element for multi-physics analysis. Multiple inheritance provides only a partial solution, allowing to inherit problem-specific capabilities from individual (single-physics) elements. The problem is that all attributes of the common parent class *Element* are either duplicated or shared (virtual base class in C++) by the derived class. Usually, the language does not provide any mechanism for selective decisions which parent class attributes are to be shared or duplicated. The use of multiple inheritance is illustrated in Fig. 1. Here, the *StructuralElement* and *Heat&MassTransportElement* classes represent problem-specific base classes. For example, all structural elements are derived from *StructuralElement* class. When an element for coupled analysis has to be developed, one naturally wants to inherit from *StructuralElement* and *Heat&MassTransportElement* classes to reuse existing implementations. However, as both parent

* Corresponding author. Tel.: +420 2 2435 4501; fax: +420 2 2431 0775.
E-mail address: borek.patzak@fsv.cvut.cz (B. Patzák).

http://dx.doi.org/10.1016/j.advengsoft.2014.01.006
0965-9978/

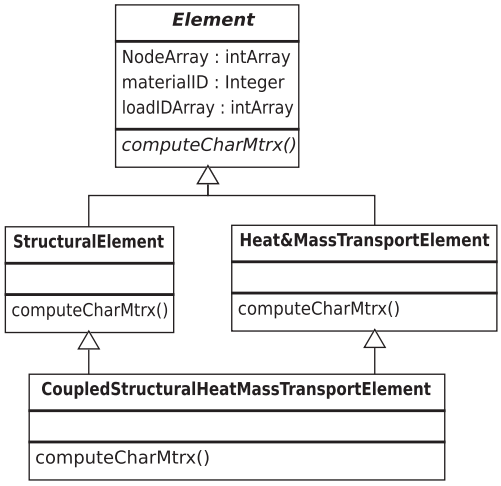

Fig. 1. Traditional approach in element definition.

classes are derived from *Element*, the attributes defined at *Element* level are either all duplicated or shared. While the geometry description should be naturally shared, the interpolation or integration description could be different for structural and heat & mass parts. Moreover, one has to manually fix name resolution problems with services defined on *Element* level.

The proposed solution consist in decoupling the common element geometry description (represented by *ElementGeometry* class) from the problem-specific functionality (represented by classes derived from *Evaluator* class). The particular element is then assembled from the base *ElementGeometry* class and a suitable *Evaluator* class. The *Evaluator* class evaluates the characteristic terms of the governing equation and is parameterized by geometry, interpolation, and integration defined by the element. The essential feature is the possibility to assemble individual *Evaluator* classes together to form a high-level evaluator for a coupled problem. Such a design allows to naturally reuse not only the evaluator for different type of problem-specific elements, but also the problem-specific evaluators when implementing a complex evaluator for a multi-physics problem.

The paper starts with an introduction to the existing design of the OOFEM code. In the next section, the proposed design supporting strongly coupled multi-physics simulations is presented. The remaining sections illustrate the implementation of coupled problems using the proposed design. The presented test cases include a typical multi-physics problem, represented by coupled heat and mass transfer in Section 5, and a damage-plastic model with implicit gradient formulation in Section 4, which illustrates how the proposed design can be exploited in mechanical analysis dealing with two independent primary unknown fields.

2. Overall design of OOFEM code

OOFEM is an object-oriented finite element framework developed at the Czech Technical University in Prague [1–3]. Its general structure is shown in Fig. 2, using the UML notation. In short, abstract classes are represented by rectangles. The lines with a triangle mark represent the generalization/specialization relation (inheritance), where the triangle points to the parent class. The lines with a diamond mark represent the whole/part relation; the diamond points to the "whole" class possessing the "part" class. Association is represented by a solid line drawn between classes. The details can be found in [4].

The *DOF* class represents a single degree of freedom (DOF). It maintains information on its physical meaning, the associated equation number, and a reference to the applied boundary and initial conditions. The base class *DofManager* represents an abstraction for an entity possessing some DOFs. It manages its DOF collection, a list of applied loads and optionally a local coordinate system. General services include methods for gathering localization numbers from the maintained DOFs, computing the applied load vector, and computing the transformation to the 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 the corresponding classes. Classes derived 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.).

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 the class derived from the *NumericalMethod* class), to solve the system of equations. The discretization of the problem domain is represented by the *Domain* class, which maintains the lists of objects representing nodes, elements, material models, boundary conditions, etc. The *Domain* class is an attribute of the *EngngModel* and, in general, 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 the matrix form, implementation is based on vector and sparse matrix representations to efficiently store components of these equations. The modular design allows uncoupling the problem formulation, the numerical solution and sparse storage being independent of each other.

3. Multi-physics design of element frame

The modular design has been achieved by decoupling the description of element geometry (represented by *ElementGeometry* class), interpolation (represented by *FEIInterpolation* class), integration (represented by *IntegrationRule* class), and evaluation of problem-specific terms (represented by *Evaluator* class). Integration rules, represented by classes derived from the base *IntegrationRule* class, define and provide a list of integration points. Derived classes represent particular integration schemes. Individual elements can have one or more integration rules; this allows to perform reduced or selected integration, or to apply integration schemes to different characteristic terms. The individual elements has to be derived from *ElementGeometry* and *Evaluator* classes and keep the list of applied boundary contitions, integration rules, interpolations, etc. For convenience, the *ElementBase* has been created, derived from *ElementGeometry* and containing a lists of applied boundary conditions, interpolations, and integration rules and keeping reference to the associated cross section and material models. These attributes are initialized by the particular element.

Element interpolation is represented by the abstract *FEIInterpolation* class, which defines general services for evaluation of interpolation (shape) functions, their derivatives, transformation Jacobians, etc. Derived classes implement a particular interpolation. The evaluation requires access to the underlying element geometry. In our approach, the element geometry is a compulsory parameter of every *FEIInterpolation* method. This allows to share a single instance of *FEIInterpolation* among all elements of the same type (static class variable in C++). Similar to the integration rule concept, elements can use several interpolations. This is essential for coupled simulation elements, where interpolation of individual fields often vary, but also allows to have different approximations for the geometry and unknown fields, for example.