Paper 255

©Civil-Comp Press, 2012
Proceedings of the Eleventh International Conference
on Computational Structures Technology,
B.H.V. Topping, (Editor),
Civil-Comp Press, Stirlingshire, Scotland

Object Oriented Design of a Coupled, Multi-Physics Finite Element Kernel

M. Horák and B. Patzák
Faculty of Civil Engineering
Czech Technical University, Prague, Czech Republic

Abstract

This paper presents the advanced object-oriented design of finite element representations in a complex multi-physics finite element environment OOFEM [1], [2]. The focus is on reuse of existing single-physics capabilities when implementing elements for coupled simulations. This has been achived 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 *ElementEvalautor*, 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 coupled analysis using implicit gradient formulation of damage-plastic model.

Keywords: multi-physics simulations, fem software design.

1 Introduction

Numerical simulations are routinely used in research and industry and are accepted as reliable analysis tools. However, in recent years, it is becoming 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 chalenges in software engineering is to design an efficient and modular modeling tools. The aim of the this contribution is to present advanced object-oriented design of general multi-physics finite element kernel allowing to reuse single physics formulations in development of coupled multi-physics problems.

The conventional designs of object-oriented finite element codes introduce an abstraction for finite element, which keep description of element geometry, properties and integration scheme and providing services for evaluating characteristic terms, such

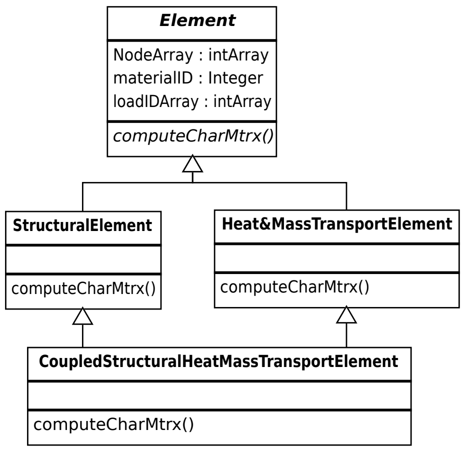

Figure 1: Traditional approach in element definition.

as stiffness matrix or element load vector. In more elaborated designs, a hierarchy of classes is developed, where base parent element class contains only problem independent description (such as element geometry) and specific functionality is implemented by derived classes, representing problem related base classes. This scheme works well when elements are to be solely used for specific analysis, e.g. elements for structural analysis.

The problem may arise, when one wants to combine capabilities of two or more elements to obtain element for multi-physics analysis. Multiple inheritance provides only a partial solution, allowing to inherit problem specific capabilities from individual (single physics) elements. This could be an issue in some programming languages (C++ for example) where the use of multiple inheritance leads to duplication of parent element class data, as illustrated in Figure. 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 classes are derived from *Element*, the attributes defined at *Element* level are duplicated. Moreover, one has to manually fix name resolution problems with services defined on *Element* level.

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

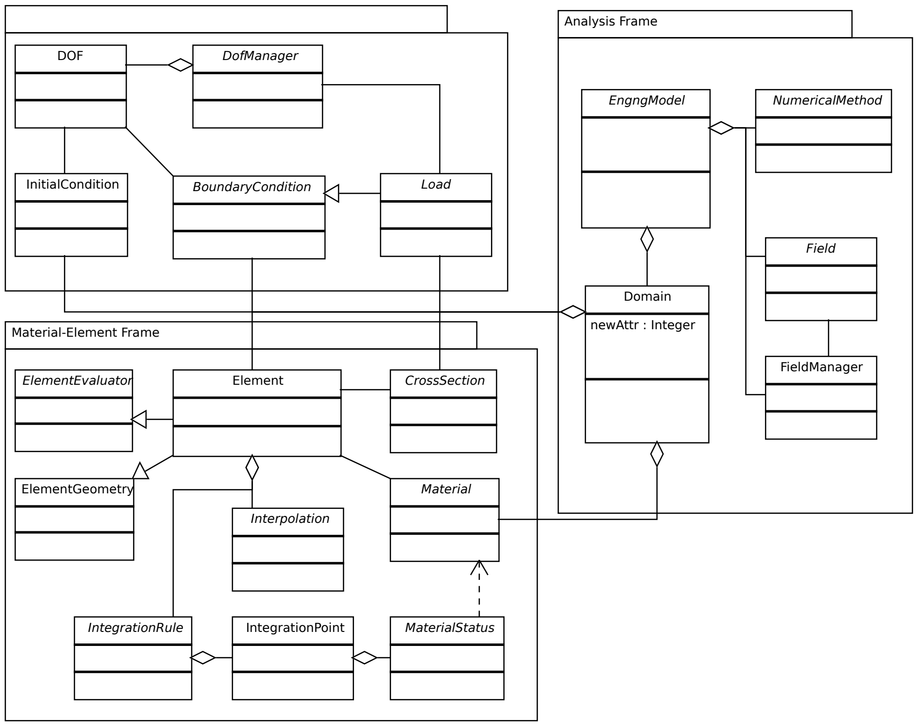

Figure 2: Overall structure of OOFEM code.

evaluators when implementing complex evaluator for multi-physics problem.

2 Overall design of OOFEM code

The general structure of the OOFEM is shown in Figure 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 line from triangle vertex points to the parent class. The lines with a diamond mark represent the whole/part relation, pointing to the "whole" class possessing the "part" class. Association is represented by a solid line, drawn between classes. The details can be found in [3].

Class *DOF* represents a single degree of freedom (DOF). It maintains its physical meaning, the associated equation number, and keeps 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, list of the applied loadings and optionally its 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 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 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, 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 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 of 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). The parent *ElementGeometry* keeps list of element nodes defining its geometry, list of applied loading, list of integration rules (defined by particular element implementation), reference to corresponding cross section and material models. Its abstract interface include methods for accessing element components, element evaluator, interpolation(s), and integration rule(s). Integration rules, represented by classes derived from base *IntegrationRule* class, define and provide 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 have individual integration schemes for different characteristic terms.

Element interpolation is represented by abstract *FEIInterpolation*, which defines general services for evaluation of interpolation (shape) functions, their derivatives, transformation Jacobians, etc. Derived classes implement particular interpolation. The evaluation requires access to underling element geometry. In our approach, the element geometry is compulsory parameter of every *FEIInterpolation* method. This allows to share a single instance of *FEIInterpolation* between 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 approximation for geometry and unknowns, for example.

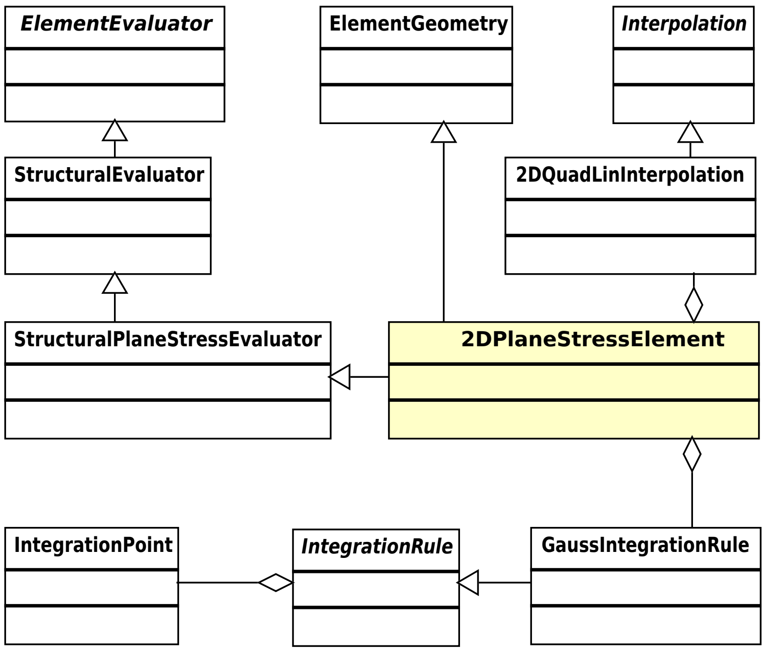

Figure 3: Collaboration diagram of *Element* class.

As already pointed out, a new abstract base *Evaluator* class has been introduced. Derived classes represent problem specific functionality of an element. Base class declares common abstract services for evaluating characteristic terms (*giveCharacteristicMatrix*, *giveCharacteristicVector*) and corresponding localization arrays. Derived classes provide corresponding implementation. The abstract interface, defined by *Evaluator*, is essential, as it allows to treat all element evaluation and assembly operations using the same general interface. As an example, consider the *StructuralAnalysisEvaluator* class implementing structural analysis functionality, see Figure 3. It provides methods for evaluation of element stiffness and mass matrices and element load vectors, based on element geometry, its interpolation and integration. Particular elements are supposed to be derived from one base *Element* class and one of classes derived from *Evaluator* class.

In coupled multi-physics simulations, one needs to combine functionality from individual sub-problems in one element (represented by corresponding classes derived from *Evaluator*) and complemented by definition of coupling terms, which will be also provided by corresponding evaluator. In order to combine individual evaluators into an evaluator for a coupled problem, the *CoupledEvaluator* class has been designed. It is derived from base *Evaluator* class and comes with the capability to group individual low-level evaluators together by performing local assembly from individual contributions. The *CoupledEvaluator* class constructor allows to set up an matrix of slave evaluators whose contributions will be assembled (locally on element level) to obtain characteristic components of coupled problem formulation. This is shown in Figure 4, illustrating mutual class relations for the case of coupled structural and

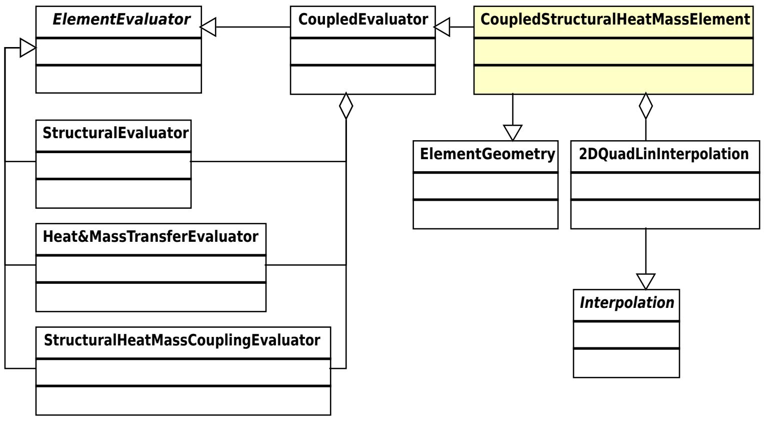

Figure 4: Collaboration diagram of *Element* in coupled analysis.

heat&mass analyses.

When problem-specific evaluator is available, the definition of particular elements is straightforward. It consist in (i) defining a new class, derived from *ElementGeometry* and class representing problem-specific evaluator, and (ii) setting up its interpolation and integration rules. No additional coding is necessary.

4 Example

In this section we demonstrate already described concept on the implementation of an implicit gradient formulation of damage-plastic model. At first, a brief description of the continuum damage mechanics and its coupling with the plasticity theory is presented, see [4] for more details. The isotropic damage behavior is considered for simplicity, which means that one single scalar damage variable is introduced. It describes the reduction of stiffness and strength of material due to the creation, coalescence and growth of voids and microcracks.

Stress-strain law has the following form:

$$\boldsymbol{\sigma} = (1-\omega)\bar{\boldsymbol{\sigma}} = (1-\omega)\boldsymbol{D}_e : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_p) \tag{1}$$

where $\boldsymbol{\sigma}$ is the nominal stress, $\bar{\boldsymbol{\sigma}}$ is the effective stress and ω is the damage variable that evolvs from zero (virgin material) to one (completely damaged material), $\boldsymbol{D}_e$ is the elastic stiffness, $\boldsymbol{\varepsilon}$ is the total strain, and $\boldsymbol{\varepsilon}_p$ is the plastic part of strain. The coupling between damage and platicity is based on formulation of the plasticity problem in the effective (i.e.undamaged) stress space.

The plastic behavior is characterized by

- yield function

$$f(\bar{\boldsymbol{\sigma}}, \kappa) = \tilde{\sigma}(\bar{\boldsymbol{\sigma}}) - \sigma_Y(\kappa) \tag{2}$$

- loading-unloading conditions

$$f(\bar{\boldsymbol{\sigma}}, \kappa) \leq 0 \qquad \dot{\lambda} \geq 0 \qquad \dot{\lambda} f(\bar{\boldsymbol{\sigma}}, \kappa) = 0, \tag{3}$$

- evolution law for the plastic part of strain

$$\dot{\boldsymbol{\varepsilon}}_p = \dot{\lambda} \frac{\partial f}{\partial \bar{\boldsymbol{\sigma}}}, \tag{4}$$

- definition of the cumulated plastic strain

$$\dot{\kappa} = \sqrt{(\dot{\boldsymbol{\varepsilon}}_p : \dot{\boldsymbol{\varepsilon}}_p)}, \tag{5}$$

Moreover, damage law has to be introduce

$$\omega = g(\kappa) \tag{6}$$

In the equations above, $\tilde{\sigma}$ is a seminorm of the stress tensor, λ is the plastic multiplier, κ is the cumulated plastic strain, σ_Y is the yield stress, and ω is the damage variable. A superior dot marks the derivative with respect to time.

Implementation of the described damage-plastic model leads to the return mapping algorithm followed by the explicit evaluation of damage.

In the regularized implicit gradient formulation, constitutive equations are enhanced by the nonlocal cumulated plastic strain which is computed from the Helmholtz equation, see [5]. Thus, its finite element implementation is based on the mixed formulation. To derive finite element formulation, we start from the strong form of the set of governing differential equations

$$\nabla \cdot [(1 - \omega(\bar{\kappa}))\boldsymbol{D}_e : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_p)] = \mathbf{0} \tag{7}$$
$$\bar{\kappa} - l^2 \nabla^2 \bar{\kappa} = \kappa \tag{8}$$

where l is the length scale parameter, ∇ is the Laplace operator, and $\bar{\kappa}$ is the nonlocal cumulated plastic strain. Note that, the nonlocal cumulated plastic strain affects only damage evolution while the yield condition remains local. Following the standard procedure, equations (7) are recasted in the weak form,

$$\int_V (\nabla \cdot \boldsymbol{\sigma}) \cdot \boldsymbol{\eta} \mathrm{d}\boldsymbol{x} \;=\; 0 \tag{9}$$
$$\int_V (\bar{\kappa} - l^2 \nabla^2 \bar{\kappa}) \zeta \mathrm{d}\boldsymbol{x} \;=\; \int_V \kappa \eta \mathrm{d}\boldsymbol{x} \tag{10}$$

where $\boldsymbol{\eta}$ and ζ are suitable test functions. The displacements and the nonlocal cumulative plastic strains are discretized at the element level by

$$\boldsymbol{u} = \boldsymbol{N}\boldsymbol{d} \qquad \bar{\kappa} = \boldsymbol{N}_{\bar{\kappa}} \boldsymbol{d}_{\bar{\kappa}} \tag{11}$$

After discretization, we obtain the set of nonlinear algebraic equations

$$\left\{ \begin{array}{c} \boldsymbol{f}_{int} \\ \\ \phi_{int} \end{array} \right\} = \left\{ \begin{array}{c} \boldsymbol{f}_{ext} \\ \\ \mathbf{0} \end{array} \right\} \tag{12}$$

in which $\boldsymbol{f}_{int} = \int_V \boldsymbol{B}^T \boldsymbol{\sigma} dx$ and $\boldsymbol{f}_{ext} = \int_{\Gamma_t} \boldsymbol{N}^T \boldsymbol{t} dS$ are the standard vectors of internal and external forces, and $\boldsymbol{\phi}_{int} = \int_V (\boldsymbol{N}_{\bar{\kappa}}^T \boldsymbol{N}_{\bar{\kappa}} \boldsymbol{d}_{\bar{\kappa}} + l^2 \boldsymbol{B}_{\bar{\kappa}}^T \boldsymbol{B}_{\bar{\kappa}} \boldsymbol{d}_{\bar{\kappa}} - \kappa \boldsymbol{N}_{\bar{\kappa}}^T) \mathrm{d}\boldsymbol{x}$ are generalized internal forces. The set of nonlinear equations (12) is solved by the Newton-Raphson method. This numerical method requires a tangent matrix, which is obtained by differentiating the internal force vector $\{\boldsymbol{f}_{int} \quad \phi_{int}\}^T$ with respect to the nodal unknowns:

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K}_{dd} & \boldsymbol{K}_{d\bar{\kappa}} \\ \\ \boldsymbol{K}_{\bar{\kappa}d} & \boldsymbol{K}_{\bar{\kappa}\bar{\kappa}} \end{bmatrix} \tag{13}$$

where

$$\boldsymbol{K}_{dd} = \int_V (1-\omega) \boldsymbol{B}^T \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{\varepsilon}} \boldsymbol{B} dx \qquad \boldsymbol{K}_{d\bar{\kappa}} = -\int_V \frac{d\omega}{d\kappa} \boldsymbol{B}^T \bar{\boldsymbol{\sigma}} \boldsymbol{N}_{\bar{\kappa}} \mathrm{d}\boldsymbol{x}$$
$$\boldsymbol{K}_{\bar{\kappa}d} = -\int_V \boldsymbol{N}_{\bar{\kappa}}^T \frac{\partial \boldsymbol{\theta}_\kappa}{\partial \boldsymbol{\varepsilon}} \boldsymbol{B} \mathrm{d}\boldsymbol{x} \qquad \boldsymbol{K}_{\bar{\kappa}\bar{\kappa}} = \int_V \left(\boldsymbol{N}_{\bar{\kappa}}^T \boldsymbol{N}_{\bar{\kappa}} + l^2 \boldsymbol{B}_{\bar{\kappa}}^T \boldsymbol{B}_{\bar{\kappa}} \right) \mathrm{d}\boldsymbol{x}$$

In the equations above, function $\boldsymbol{\theta}$ maps the strain at the end of step $\boldsymbol{\varepsilon}^{n+1}$ onto the stress at the end of step $\boldsymbol{\sigma}^{n+1}$ and function $\boldsymbol{\theta}_\kappa$ maps the strain at the end of step $\boldsymbol{\varepsilon}^{n+1}$ onto the cumulated plastic strain at the end of step κ^{n+1}. Both functions are supplied by the return mapping algorithm. Matricies $\boldsymbol{B}$ and $\boldsymbol{B}_{\bar{\kappa}}$ contains derivatives of the shape functions.

4.1 Implementation

To implement present model within the structure described in previous chapters, four *Evaluator* classes have to be combined. The first one, *Structural Evaluator*, evaluates $\boldsymbol{K}_{dd}$, $\boldsymbol{f}_{int}$, and $\boldsymbol{f}_{ext}$ which corresponds to the classical stiffness matrix, internal forces vector, and external forces vector. The second one, *ImplicitGradient Evaluator*, evaluator evaluates terms $\boldsymbol{K}_{\bar{\kappa}\bar{\kappa}}$, ϕ_{int}, and ϕ_{ext} which are related to the Helmholtz equation. Remaning two evaluators provide coupling terms $\boldsymbol{K}_{d\bar{\kappa}}$ and $\boldsymbol{K}_{\bar{\kappa}d}$. The *Coupled Evaluator* is set up using a matrix of individual sub-problem evaluators. This matrix naturally maps the individual terms in problem Jacobian (13) to the corresponding Evaluator instances and also the characteristic vectors of the problem (12) to the evaluators on the diagonal of evaluators matrix.

The set up of *Element* and corresponding *CoupledEvalauator* is demonstrated in the code listing, see Listings 1 and 2. In this example, a new *MyElement* class is defined with capabilities required by above described problem. The objects representing displacement and gradiend field interpolations, as well as sub problem evaluators are

```
// declaration of new "MyElement" class
class MyElement : public ElementGeometry,
      public CoupledEvaluator {
public:
  // declare individual interpolations
  // as static (class) variables

  // displacement interpolation,
  static FEI2dQuadLin dipl_interp;
  // gradient interpolation
  static FEI2dQuadQuad grad_interp;

  // sub-problem evaluators, common to all
  // instances of MyElement -> static variable
  static StructuralEvaluator E_dd;
  static HelmholzEvaluator E_kk;
  static StructuralHelmholzEvaluator E_dk;
  static HelmholzStructuralEvaluator Ekd;

  MyElement (int n, Domain* d);
};
```

Listing 1: MyElement declaration

```
// implementation
FEI2dQuadLin MyElement::dipl_interp ();
FEI2dQuadQuad MyElement::grad_interp ();
// set up individual sub-problem evaluators
StructuralEvaluator MyElement::E_dd (1);
HelmholzEvaluator MyElement::E_kk (2);
StructuralHelmholzEvaluator MyElement::E_dk(1,2);
HelmholzStructuralEvaluator MyElement::E_kd(2,1);

// set up matrix of sub-problem evaluators
static Evaluator eMatrix[2][2]=
  {{&MyElement::E_dd, &MyElement::E_dk},
   {&MyElement::E_kd, &MyElement::E_kk}};


// Constructor
MyElement::MyElement(int n, Domain* d):
  ElementGeometry(n,d), CoupledEvaluator(eMatrix)
{
  // set up integration scheme
  numberOfIntegrationRules = 1;
  integrationRulesArray =
    new IntegrationRule * [ 1 ];
  integrationRulesArray [ 0 ] =
    new GaussIntegrationRule(1, this);
}
```

Listing 2: MyElement class

declared as static class variables. These are associated with the class itself rather than with any class object and every instance of a class shares the same class variable(s). The *MyElement* class constructor demonstrates also initialization of parent coupled evaluator class using a matrix of sub-problem evaluators instances. In the constructor, the element integration rules are set up, in this case each instance of new element creates its own copy of integration rules, as the integration point and associated load-time history internal variables are unique for each element. This concludes the element definition, and in principle, no additional coding is necessary when introducing a new element as all the functinality is provided by parent *ElementGeometry* and problem related *ElementEvaluator* class. The role of element definition is just to assemble together right combination of evaluator capabilities, interpolations, and integration rules.

4.2 Biaxial compression test

The regularization capabilities of the implicit gradient model are explored using the classical biaxial compression test [6] with hardening Mises plasticity coupled with isotropic damage, see Figure 5 for the problem statement. The problem is modeled as a plaine-strain problem with material and geometrical parameters summarized in Table 1.

Height of the specimen	L	120 mm
Width of the specime	B	60 mm
Young modulus	E	20 GPa
Poisson ration	ν	0.2
Isotropic hardening law	$\sigma_Y = \sigma_0 + H\kappa$	
Initial yield stress	σ_0	100 MPa
Hardening modulus	H	400 MPa
Damage law	$\omega = 1 - \exp^{-a\bar{\kappa}}$	
Dimensionless damage parameter	a	30
Characteristic length	l	5 mm

Table 1: Biaxial compression test: Geometry and material parameters

The problem was discretized by a quadrilateral finite elements with a quadratic interpolations of the dispalcement and a linear interpolation of the cumulated plastic strain. The mesh insensitivity is illustrated by results on two different meshes with a constant imperfection of size 10 mm x 10 mm, see Figure 6 and Figure 7.

5 Conclusions

The advanced object-oriented design of finite element representations in a complex multi-physics finite element environment has been presented. It allows to reuse of

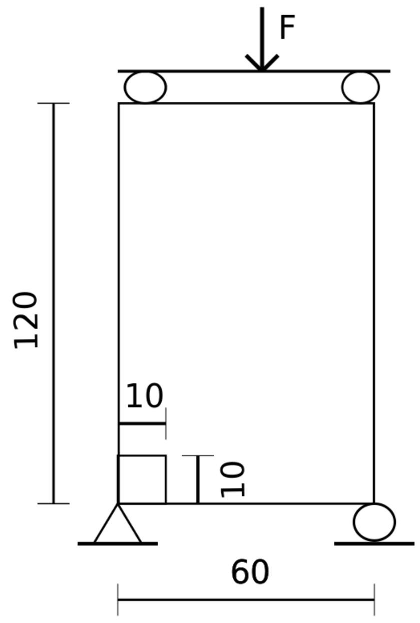

Figure 5: Biaxial Compression Test: Geometry and Loading

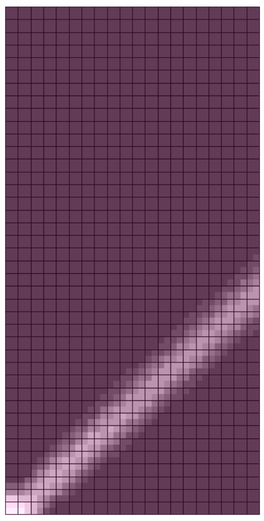

Figure 6: 20x60 elements

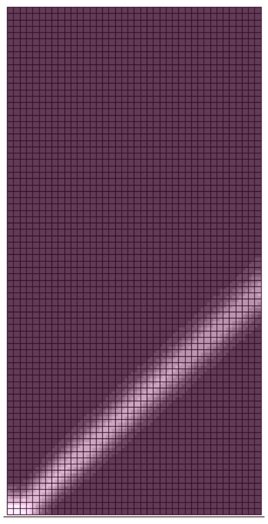

Figure 7: 40x120 elements

Figure 8: Biaxial Compression Test: Damage Patterns

existing single-physics capabilities when implementing elements for coupled simulations. This has been achived 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 *ElementEvalautor*, can be assembled together to define an evaluator for coupled analysis. Presented design leads to extremely flexible implementation, with clean modular design. The application has been demonstrated for a coupled analysis using an implicit gradient formulation of damage-plastic model.

Acknowlegements

This work has been supported by the Grant agency of the Czech Republic, projects no. P105/10/1402 and no. GAP108/11/1243.

References

[1] B. Patzák and Z. Bittnar, Design of object oriented finite element code. Advances in Engineering Software, 32(10-11):759–767, 2001.

[2] B. Patzák, OOFEM project home page, http://www.oofem.org, 2012.

[3] M. Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley, ISBN 0-321-19368-7, 2003.

[4] G. Maugin : The Thermomechanics of Plasticity and Fracture, Cambridge University Press, ISBN 0521397804, 1992.

[5] R. De Borst, L.J. Sluys, H.B. Muhlhaus, and J. Pamin, Fundamental issues in finite element analyses of localization of deformation, Engineering Computations 10, 99 – 121, (1993)

[6] R. de Borst and J. Pamin, Some novel developments in finite element procedures for gradient-dependent plasticity, International Journal for Numerical Methods in Engineering, 39: 2477 - 2505, 1996.