USAGE OF MODULAR SCISSORS IN THE IMPLEMENTATION OF FEM[*]

Dalibor Frydrych

Abstract

Finite Element Method (*FEM*) is often perceived as a unique and compact programming subject. Despite the fact that many *FEM* implementations mention the Object Oriented Approach (*OOA*), this approach is used completely, only in minority of cases in most real-life situations. For example, one of building stones of OOA, the interface-based polymorphism, is used only rarely.

This article is focusing on the design reuse and at the same time it gives a complex view on *FEM*. The article defines basic principles of *OOA* and their use in *FEM* implementation. Using OOA *FEM* project is split in many smaller sub-projects which are interlinked together. Links between sub-projects are one way only and non circular. Such a setting gives opportunity to use the modular scissors. In addition, these individual sub-projects can be used directly, without additional adjustments, in similar projects.

1 Introduction

Development of computer programs have undergone significant changes in recent years. In the beginning, programming was seen as a kind of art. Programmers worked alone and quality of their work depended on their individual skills. The demand for computer programs increased significantly with rising use of computers. At the same time expectation from these programs were growing too, in parallel with the complexity of studied tasks. A new market was created and new companies emerged to develop computer programs. Due to complexity of tasks programs were usually written by several programmers. This brought along approach known, up to that date, mainly in mass production: analysis and planning standardization, quality assurance, process documentation, personal replaceability, labor efficiency and management. Development of computer programs became a full scale industry.

Process of program development is structured like other projects. Development is done by a development team. Team is managed by Project manager. A task is initially analyzed, if necessary also in depth at customer side, by an analyst. Designer is supposed to choose an appropriate software platform and language. Programmer (SW designer) is often called a "coder" and is responsible solely to write machine code using data input defined by analyst and designer.

[*]This work has been supported by Ministry of Education of the Czech Republic; under the project "Advanced remedial technologies and processes", code 1M0554.

Organizing the project development in different layers between different persons improves productivity. Team member on given level is an expert in his job. Communication between the layers is standardized by using defined documents (equivalent of technological procedures for example in industrial production). Definition of all these steps ensures personal replaceability and makes team management easier. In order to ensure that documentation is consistent in all levels it was necessary to adjust general approach to problem solving. Nowadays, the ideal approach seem to be Object Oriented Approach (*OOA*).

2 Object oriented approach

Common mistake committed by developers is, to use for solving of a problem the object oriented programming language, and to believe that it is *OOA*. *OOA* is mainly about giving a project right structure, than about using particular programming language. For example, one of important features, by which *OOA* can be recognized is the absence of circular associations.

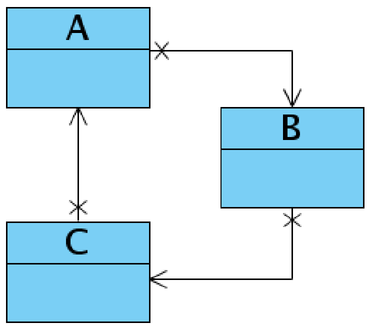

Fig. 1: *Class diagram of circular association.*

Circular associations are often responsible for tangling of code. Figure 1 demonstrates circular association between three classes **A**, **B** and **C**. Class diagram from Unified Modeling Language (*UML*) [2] is used for explanation. Modeling of processes, where classes are so heavily interlinked is very complicated as change in one class causes changes in other classes. During implementation, many such conflicts arise and what makes solving of problem even more difficult is the fact, that responsibility of individual classes is not clearly defined.

Solution to this problem is to cut the circular association, for example between class **C** and **A**, and make it linear. Then the responsibilities of classes are clearly defined. Implementation of class **C** is the starting point. Because class **C** is not dependent on any other class, its responsibility is clearly defined. Implementation is quick and easy. Next step is implementation of class **B**. This class is using class **C**. But class **C** is already implemented and tested, so it is ready to use. The same situation repeats in implementing of class **A**.

However, decision how to cut circular association is a crucial step and has to be done only after deep analysis of the whole system.

3 Analytical model of FEM

Now, it is time to describe *FEM*, very roughly and analytically, from system design point of view. Domain of task Ω is divided in particular sets of elements. Scalar products are calculated for each element. Values of individual scalar sets members depend on initial conditions, respectively on results of previous time step. Scalar products are placed in global matrix. Global matrix and the right hand side of linear equations set are modified according to boundary conditions. Set of linear equations is solved, and then interpreted as values of individual searched parameters. Then the calculation is repeated for next time step.

From the above description it is clear, that the *FEM* is element-centric method. In many implementations of *FEM*, the data structure defining an element is very big.

```
class Element {
   Node[] nodes;
   ...
   MaterialParameters[] materialParameters;
   DOF[] dofs;
   Results[] results;
   ...
}
```

It contains information about initial conditions, material properties, boundary conditions etc. Such a structure is too complicated and difficult to re-use. Defining too big structures is very common design mistake.

3.1 Reduced model of mesh

The key point of efficient design is making the data structure, which defines the element, smaller. The basic idea is the following: element defines only part of the task domain Ω, an element needs only association to nodes.

Analytical model of mesh is then very simple, as can be seen in Figure 2. Mesh keeps only information about task domain Ω, ensures the association to list of ele-

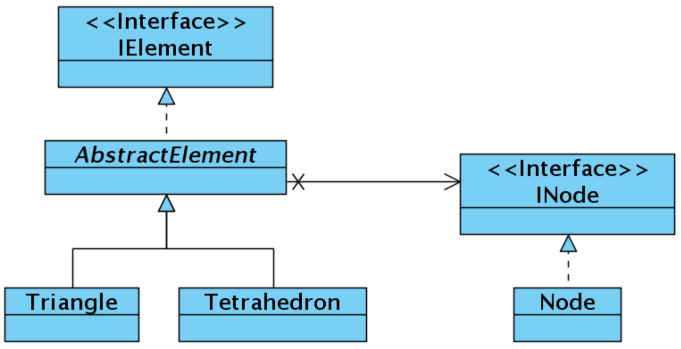

Fig. 2: *UML class diagram of mesh.*

ments and list of nodes. An element is a typical abstract structure. Mechanism of inheritance is used to specify concrete type of element (triangle, tetrahedron, ...). A model defined in this way has a clearly defined functionality, which can be used in future models with no need for additional modification - that is the idea of re-use.

3.2 Methodology DF^2EM

Methodology DF^2EM (Developers Fab Finite Element Method) [5] describes individual particular models, their functionality and associations in between them, see Figure 3.

The lowest level represented by model *Tools* is dealing with system functions. The next level model *Math* is solving main mathematical parts. All these models work only with basic sets of data (int and double), eventually define their own (*Matrix*). On the next level there are three independent models:

- Model *Mesh* described above.
- Model *Material* implementing calculation with material parameters.
- Model *Scenario* solving definition of individual time steps for calculation of unsteady processes.

Model *Approx* is a database of approximation functions. Model *Local* interconnects, using association classes models *Mesh*, *Material* and *Approximation* and implement calculations of local matrices. Model *Formulation* defines basis of mathematics formulation of *FEM* (primal, mixed-hybrid, etc.). On the highest level there is model *Task*. It ensures initial phases, start up and management of the whole calculation. From Figure 3 it is clear that associations between individual models are NOT circular. To re-use any individual model, it is possible to use modular scissors. For example, to re-use model *Local*, it is necessary to take ONLY models on which this model is dependent (*Approx*, *Material*, *Mesh*, etc.) NOT all the models *Methodology* DF^2EM.

4 Implementation

Programming language *JAVA* was used for implementation. Implementation was named as *Project* DF^2EM and was based on models created in *Methodology* DF^2EM. Interface-based polymorphism was exclusively used for implementation in order to ensure easy exchange of concrete implementation. *JAVA* language ensures the possibility to insert concrete implementation even in run-time (similar to principle of plug-in modules). User of *Project* DF^2EM has therefore an option to exchange any part of implementation by a different one without need to modify the source texts *Project* DF^2EM.

4.1 Testing

An important part of each software development project is its testing. For implementation of *Project* DF^2EM were used ideas of technology *Test driven development*. Division of *FEM* into small logically defined parts, enabled their thorough

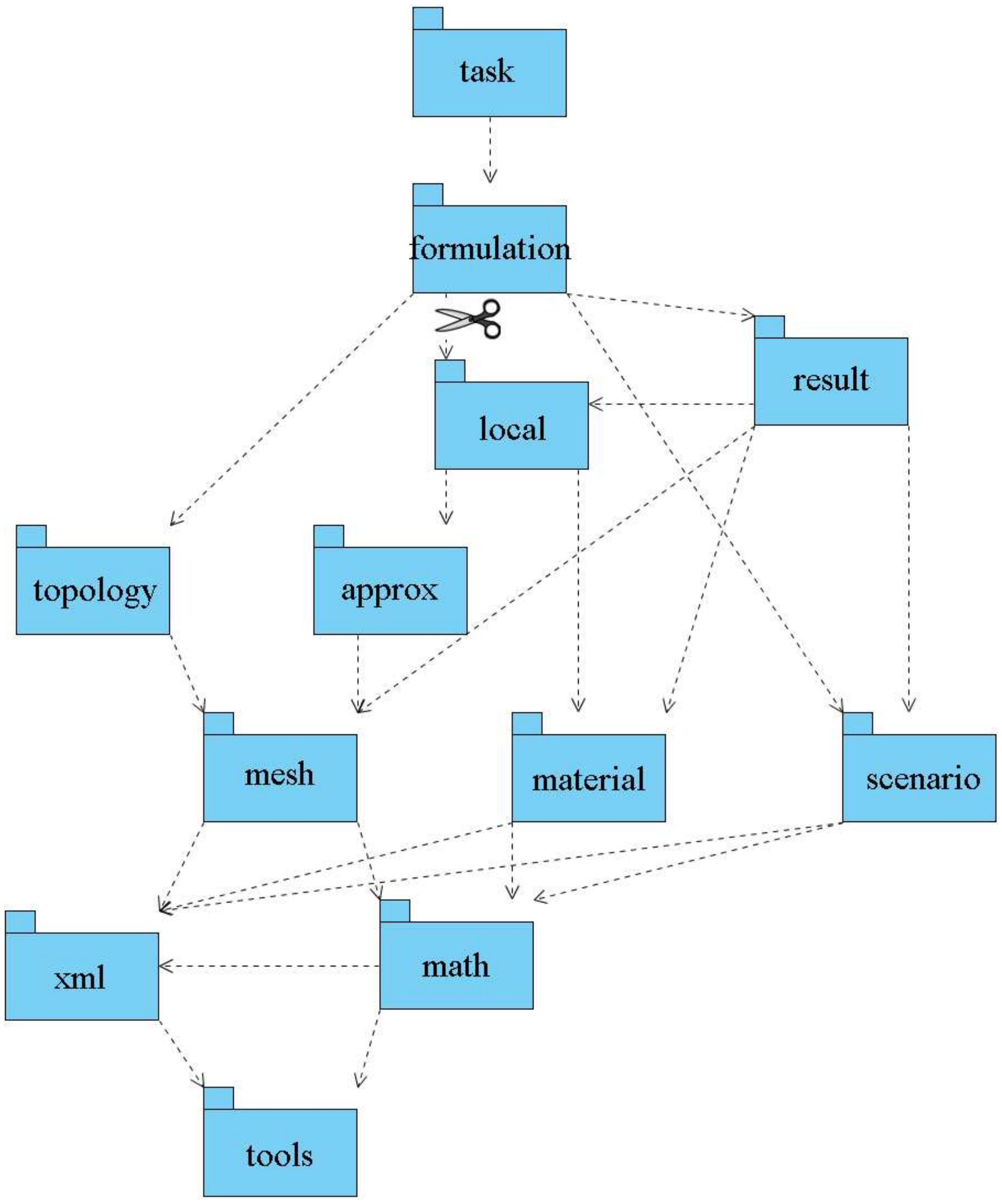

Fig. 3: *Schema of* DF^2EM *models.*

testing. Framework *JUnit* [3] was used for testing. Thorough testing was possible by using support tool *Cobertura*. In perimeter of *JUnit* it is possible to create test file for each class. Then it is possible, within the class, to test all its methods. Tool *Cobertura* analyzes how all individual tests were running. Checks track of all tests passages through individual lines of source code and presents results in graphs and reports. In this way it enables very detailed testing of all the functions of individual classes.

5 Conclusion

This article defined a unique system of *OOA* to *FEM* called *Methodology* DF^2EM. Basis is *Methodology* DF^2EM detailed *OOA* of *FEM*. This *OOA* is independent of used programming language. *FEM* is divided in small, logically defined modules - which are managing individual objects. Associations between modules were reduced so between them and also inside of them circular associations were not created. Avoidance of circular associations, allow as to separate individual modules for re-use, by modular scissors. Such models can be re-used in other project without additional corrections or adjustments.

Methodology DF^2EM is not only a theoretical work with no practical use. Implementation was done in language *JAVA* and is called *Project* DF^2EM. *Project* DF^2EM is used for implementation of several models based on *FEM*. Very good results were reached in model *ISERIT* [6].

References

[1] Gamma, E., Helm, R., Johnson, R., and Vlissides, J.: *Design patterns: elements of reusable object-oriented software*. Addison Wesley Professional, 1994, ISBN 978-0201633610.

[2] UML©Resource Page, `http://www.uml.org`

[3] JUnit.org: Resources for Test Driven Development, `http://www.junit.org`

[4] Cobertura, `http://cobertura.sourceforge.net`

[5] Frydrych, D. and Lisal, J.: Introduction to methodology DF^2EM - framework for efficient development of finite element based models. *Proceedings of ICCSA'08*. San Francisco, USA, 2008.

[6] Frydrych, D. and Hokr, M.: Verification of coupled heat and mass transfer model ISERIT by full-scale experiment. *Proceedings of ICMSC'08*. San Francisco, USA, 2008.