Introduction to *Methodology DF^2EM* - Framework for Efficient Development of Finite Element Based Models

D. Frydrych [*] and J. Lisal [*]

Abstract— **The Finite Element Method (*FEM*) is currently one of the most widely used methods to solve engineering questions described by partial differential equations. This method has been implemented in several commercial software packages. These are in most cases proprietary systems, whose architecture and source code are closely guarded. Their further development, customization or adding of new functionality is quite time consuming, difficult and costly.**

This article presents a different approach to development and use of *FEM* based models. It presents *Methodology DF^2EM* which is based on the idea of open source software. It is an open system, which enables easy and fast development of new model systems based on *FEM* and it also defines standard interfaces based on which it is possible to develop own parts of the model, compatible with *Methodology DF^2EM*.

Keywords: mathematical modelling, OOP, UML, XML, JAVA, design pattern

1 Introduction

The development of modern software applications does not depend solely on the writing of the source code. The development takes place in a world of fast development of requirements, frequent change requests, new tasks, aggressive deadlines and constant cost pressure. In order to succeed the team leader, the SW analyst, programmer and the entire development team as such must pay close attention to proper analysis of the task at hand, requirement specification, detailed solution design documentation and optimal architecture design of the future system. This approach is already commonplace in the software industry, its utilization in the academic circles is however lagging behind the commercial world. This is probably caused by the perception that academic SW projects are generally smaller and their documentation does not add much value.

Methodology DF^2EM is much more than a general description of the steps and definition of interfaces. Its inseparable part is the implementation of those interfaces in particular classes. These classes form a framework within which you can develop concrete models. The user then uses existing source code when developing their own model and only concentrates on enhancing the functionality. The benefits of Object-Oriented Programming (*OOP*) are exploited.

When designing *Methodology DF^2EM*, great care was taken to address structuring of individual parts (subprojects). This division ensured high reusability of already developed pieces and provides for an easy deployment of those subprojects in other projects. During its development the most advanced technologies and methodologies have been used from the areas of design analysis, system design and implementation.

This article presents new directions in software development and their impact on the development of mathematical models. It demonstrates the advantage of documented, well managed development applied in the academic circles.

1.1 *FEM* from *OOP* point of view

Classical schema describing the general approach to *OOP* is based on division in two Meta-layers.

$$\text{Real thing} \rightarrow \text{class} \rightarrow \text{meta class}$$

The first Meta-layer deals with the transformation of real world task in the computer environment by describing its characteristic properties and behavior. The product of this layer is a class.

[*]Technical University of Liberec, Faculty of Mechatronics and Interdisciplinary Engineering Studies, Institute of new technologies and applied informatics, Studentská 2, 461 17 Liberec, CZECH REPUBLIC, EUROPE, Emails: ffast_train@quick.cz, jan.lisal.frisco@topdance.cz

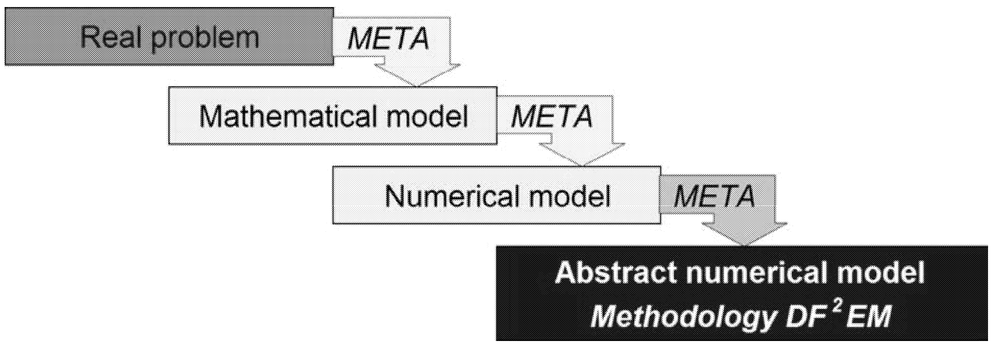

Figure 1: Structure of Meta-layers

The second layer is more complex. It generalizes the properties and behavior of various classes. The product of this layer is a Meta-class.

When working with mathematical models, the above-described schema is extended by another Meta-layer, see figure 1. The classes no longer describe particular real-world task, but rather its abstract representation in the form of mathematical equations. This significantly complicates the task at hand. Detailed documentation becomes necessary to keep track of the entire project. The first level of documentation is the analytical model.

2 Analytical model

The framework of calculation based on *FEM* can be divided in the following steps:

1. Loading an init-file of the **model** and loading an input data of the **task**. *XML* format has proven very useful, see section 4.
2. * Generating of **approximation_functions** and **test_functions** (depends on the *FEM* formulation being used). A **function** from an extensive library of frequently used **functions** can be used (for example **polynoms**) or this library can be extended with new **functions**.
3. * Creation of structures of the **global matrix**, the **right_hand_side_vector** and the corresponding **access_vectors** to gain access to the **individual_items** of **global_matrix** and of the **right_hand_side_vector** to the **entities** of space discretization (depends on the **task** at hand and on the **formulation** of the *FEM* being used).
4. Initialization of **calculation_scenario**, or the start of the next **step** (of **calculation_scenario**).
5. Calculation of **local_matrixes**. In case the **local_matrix** does not change during the **calculation** (constant material properties, fixed mesh) the values of the items of the **local_matrixes** are calculated only once, in the first **step** of the **calculation_scenario**.
6. Placement of the **local_matrixes** into the **global_matrix** and the assembly of the **right_hand_side_vector**.
7. * Application of **boundary_conditions** in the **global_matrix** and the **right_hand_side_vector**.
8. Solving the **set_of_linear_equations** defined by the **global_matrix** and the **right_hand_side_vector**.
9. * Distribution of the **solution** of the **set_of_linear_equations** into the appropriate **entities** of space discretization. The values can be calculated directly in **entities** where the *FEM* does not provide the solution directly (for example dual values).
10. Evaluation of the success of the **step**, set up of stop-criteria. For example the quality of the solution can be improved through finer discretization - by using **adaptive_mesh**.
11. Repeating of a given calculation **step** in case the previous **step** didn't finish successfully (back to point 5).
12. Next **step** of the **calculation_scenario** (back to point 4).
13. **Results** output. General **results** usable for graphical postprocessing are meant here.
14. * Individual results **processing**. This concerns special, user-defined calculations, for example:
    (a) Time development of a given value in given point in space. Interpolation using appropriate **approximation_function** given by the used **formulation** *FEM* is used for this **calculation**.

(b) Integral criterion for a given value in the area to be solved (for example tension energy for elasticity task, or the volume of water in tasks regarding filtered water flow).

Names written in bold are candidates to become a class. The internal structure of the classes is not considered on this level. The goal is to obtain general overview of the system being developed.

From the textual analysis we can extract high percentage of functionalities (marked with an asterisk) which can be further generalized. These generalized functionalities are characteristic for any numerical model based on *FEM*. The proportion of functionalities specific for a given *FEM* task is fairly small. The items covering these functionalities are marked with an asterisk.

Creation of a new model based on *FEM* can be reduced with the utilization of $Methodology\ DF^2EM$ to the creation of a few classes, whose interface and interaction with the environment is defined by $Methodology\ DF^2EM$.

3 UML diagrams

Unified Modeling Language (*UML*) [4] is successfully used to clearly capture the requirements for the newly designed system. *UML* provides graphical expression of design thoughts through the use of standardized diagrams. Class diagram is the most useful of the *UML* provided tools for mathematical modeling. Graphical expression of the relations between classes makes it possible even for non-developers to understand the structure of the numerical model.

Class diagram describing a package was chosen to present an example of utilization of *UML* in $Methodology\ DF^2EM$. This package describes discretization of problem domain, which is generally called mesh, see figure 2.

The existence of circular relationships is eliminated even on this basic level. Each element is defined by appropriate nodes, from which it obtains information about its coordinates. On the other hand, a node can be shared by several elements so that it can inform them in case it moves. This circular reference was eliminated by the use of design patter called "Observer", see chapter 5.1.

UML has proven to be a means of effective communication between experts from different fields in this project.

4 *XML* input files

Reading of input data from a file has proven to be quite problematic. It was necessary to ensure high variability while maintaining maximum possible independence. Detail documentation and certain level of standardization were further required.

Extensible Markup Language (*XML*) [5] format was eventually chosen for the input data. Individual parts of the calculation and the appropriate classes are grouped into logical sections. This arrangement provides for a high flexibility when preparing input data, for example for "variant" calculations. Changes in one section thus do not affect other sections. The nesting can be used, where the structure of one internal section is used in more sections. $Methodology\ DF^2EM$ defines two types of input files.

4.1 Initialization file

This file defines classes, from which instances are created, which are further inserted into the basic framework. Initialization file ensures the model initialization for solving a specific task type (heat task, elasticiy taks, etc.).

4.2 Input data file

Input data file contains data defining a specific task at hand. The reading of the input file is divided into these steps:

- Reading of discretization of the solutin domain (mesh). The model can work with various systems for mesh generation. The task of discretization of the problem domain is solved by external systems and $Methodology\ DF^2EM$ is responsible for reading the files generated by these systems.
- Reading of material properties. Individual material properties are distinguished by title (abbreviation). The file with material properties can therefore be used for calculation of various tasks where a particular model only uses the properties needed for the modeled process and the remaining properties are ignored. All properties are treated as functions, dependent on target variable. Each task is treated as non-liner task. Definition of functions is provided in text format, which is processed by function parser, section 5.2.
- Reading of boundary conditions. Each task is considered to be an unsteady task. The action is described by calculation scenario. The calculation scenario is based on several regimes of boundary conditions. The duration and the number of equidistant steps is given for each regime. The regime is further based on provided boundary conditions. If the first step has zero duration and zero steps then it is solved as steady task. The solution then corresponds to the initial conditions, which are in this case not read (the following step will be skipped).

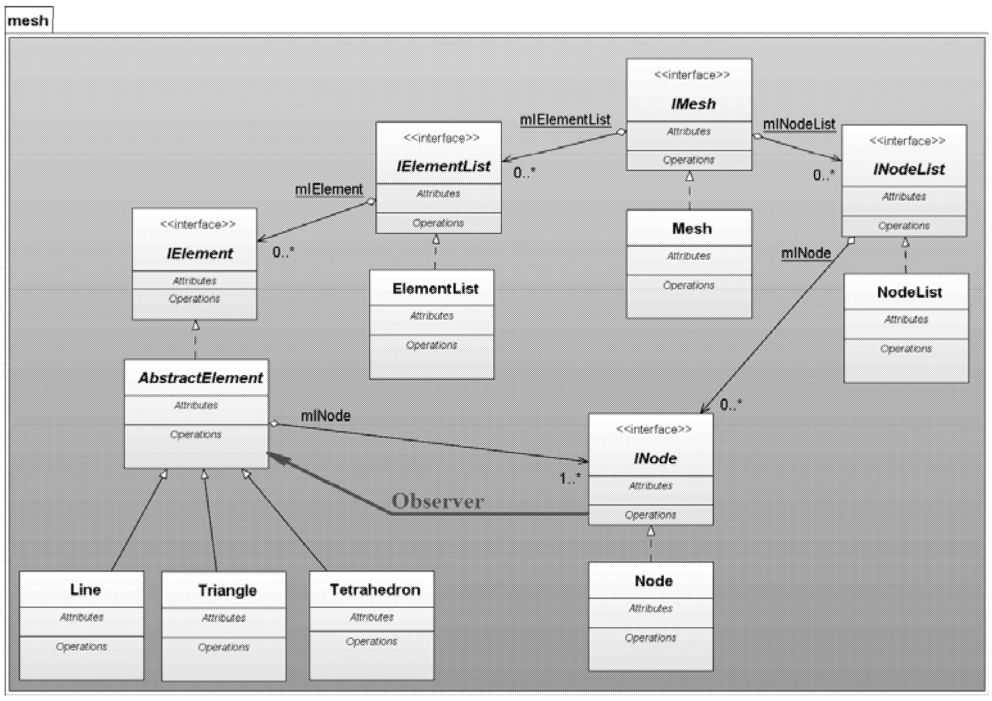

Figure 2: Class diagram of package mesh

The values of the boundary conditions can be specified:

– Constant for the entire duration of a regime in which they are defined.

– The value at the beginning and at the end of regime, where the values in between are interpolated.

– Lexical analyzer is used to define the boundary condition (in addition to other options TT - time since the beginning of the task or TR - time since the beginning of the regime can be used)

The actual values are read from the appropriate section of the *XML* file.

- Reading of the initial conditions. In case of solving an unsteady task with given initial conditions, those conditions are read from the appropriate file. The input file for initial conditions is compatible with the output file. This guarantees, that the output data from one task can be used as input for some other task (task branching).

XML has proven its usefulness in this project, as it ensured general usability of output files and effective work with them.

5 Implementation

The analysis and design of the system have been, so far, platform and language independent. During the selection of the platform and programming language, the following requirements have been taken into account:

- Easy portability between various platforms and operating systems.
- Stable and portable, object oriented programming language, which helps minimize human errors.
- Low cost.
- Usability in education.

5.1 *JAVA* language

Based on the above-mentioned requirements, *JAVA* has been selected as the programming language of choice [1]. It is necessary to note, that the basic requirements did not include performance or speed of execution. This is a result of the fact that the speed of execution of the resulting program is not significant when considering the total amount of time needed to develop the model. $Methodology\ DF^2EM$ is a tool for rapid development of new models and testing of their properties. The speed of execution of *JAVA* is not a limiting factor even for more extensive tasks then the ones at hand. *JAVA* can be considered an ideal programming language for projects similar to $Methodology\ DF^2EM$.

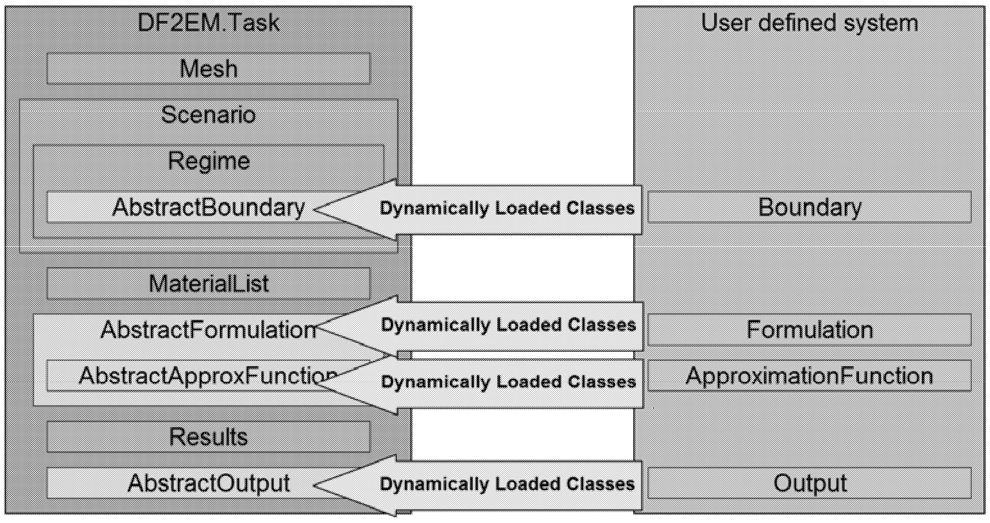

Figure 3: Scheme of using the dynamic loading of classes in *Methodology* DF^2EM

Another framework that has been successfully used in the implementation are design patterns [3]. The "Interpreter" pattern has been used for the implementation of "automatic" updates. To instantiate all classes, the "Factory method" is used consistently. The **Class.forName** technology provided within the *JAVA* language is used in the developed software, see figure 3. Its use extends the "Factory method" to the level of "Abstract factory".

5.2 Parser of functions

The implementation of parser of functions has proven very useful. Full flexibility has been achieved for entering values in the input files in this way. The value does not need to be limited to a constant (when using function parser, a constant function can be used) but rather any function can be used. The user therefore gains the flexibility to easily introduce nonlinearities (for example the values of material properties, boundary conditions, etc) without changing the source code. At the same time, the user needs to keep in mind that the task of convergence and uniqueness of solution remains with him/her.

The actual implementation of the function parser has been divided into three parts:

- The first step is the implementation of interface *IFunction*. This interface is quite simple and contains only one method *double value(double[] variables)*.
- The second step is the implementation of abstract method *AbstractFunctionParser*, which realizes the interface *IFunction*.
- The third part is the implementation of a parser for a particular function. This is, in fact, an implementation of a single method (constructor) which defines the internal order of independent variables.

5.3 Used software tools

An integral part of implementation is information about tools used for implementation. The following list summarizes the tools used and their brief description. Some of the names bellow are a registered trademarks.

Java - programming language (version 1.6).

NetBeans - Integrated Development Environment (IDE 6.1 release) for software developers. It enables model development starting from *UML* all the way to implementation in *JAVA*.

Subversion - centralized system for sharing source code files. The core of system is a repository, which acts as central store of data. Any number of clients can connect to the repository, and then read or write to these files.

Maven - software project management and comprehension tool. Maven is based on the concept of a project object model (POM). Maven manages a project build, reporting and documentation.

Cobertura - tool, that calculates the percentage of code accessed by tests. It is used to identify which parts of your *JAVA* program are lacking test coverage.

6 Conclusions

Methodology DF^2EM is the multi-META-layer system, see figure 1. It can be used as an end-user tool to build, from the provided components, own modeling systems to solve concrete tasks. For example to define own material parameters (using function parser - see section 5). In this case the work takes place on the first META-layer. These type of changes do not require programming knowledge from the user (creation of new classes). Therefore these changes can be implemented quickly.

The second META-layer represents changes to the mathematical model and their impact on the numerical model. It could be, for example, definition of new approximation functions, or the use of a new solver for a set of linear equations. This layer requires basic programming skills. The user is guided by technical documentation. Great benefit can be obtained from studying classes which are part of $Methodology\ DF^2EM$.

The third META-layer is intended for users with very good knowledge of $Methodology\ DF^2EM$. On this layer, the user can make changes to the basic framework.

An integral part of $Methodology\ DF^2EM$ are tests. For each class there is a class that provides testing of its functionality (for example stability when changing internal state). There are also classes responsible for testing larger functional areas (for example the compilation of approximation functions). The highest level of tests is represented by functions which test sample tasks (for example evaluation of results of a particular task in a given mesh, material and given boundary conditions).

$Methodology\ DF^2EM$ is not an artificial academic system. Several other systems have been building on top of the provided framework, which were used to solve concrete problems. Among the most successful and most widely used belongs model *ISERIT* [2]. The *ISERIT* solve the unsteady coupled heat and moisture transfer problem in porous materials with sorption of moisture in solid phase by using primal formulation of finite element method. During the implementation of the *ISERIT* model, it was necessary to implement only 24 classes (logically divided into 6 package) out of which 12 classes deal with boundary conditions. The implementation of the *ISERIT* model took about 3 weeks. Significantly more time was needed to prepare input data [2].

$Methodology\ DF^2EM$ has proven its validity and is used for the development of new models dealing with tasks being solved at the Institute of new technologies and applied informatics on Technical University of Liberec.

Acknowledgements

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

References

[1] Eckel, B., *Thinking in JAVA*, [online] URL:<http://www.bruceeckel.com>

[2] Frydrych, D., Hokr, M., "Verification of Coupled Heat and Mass Transfer Model ISERIT by Full-scale Experiment" *ICMSC'08 - International Conference on Modeling, Simulation and Control*, San Francisco, USA, 10/2008

[3] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software* Addison Wesley Professional, 1994, ISBN 978-0201633610

[4] Pilone, D., Pitman, N., *UML 2.0 in a Nutshell*, First Edition, O'Reilly Media, Inc., 2005, ISBN 9780596007959

[5] Simpson, J., E., *Just XML*, Prentice Hall PTR, 2000, ISBN 013018554X