Object Oriented Design of Field Analysis Translator

Piotr Rowiński, Jacek Starzyński, Stanisław Wincenciak

Abstract— **The paper presents object oriented approach for design of the software package for electromagnetic fields simulation. The most important stages of the objective software design are discussed.**

I. Introduction

THE first idea of the problem oriented language for finite element based electromagnetic fields simulator was born in the Department of the Theory of Electrical Engineering (currently Department of The Electrical Engineering and Applied Informatics) of Warsaw University of Technology in 1985. The concept of problem oriented language was borrowed from the circuit simulators as SPICE and NAP. The authors wanted to design a similar computer language for formal description of 2D boundary problems and tasks of field analysis, optimal design and identification.

Following several test releases of the language and its interpreter the first stable version—Field Analysis Translator (FAT) 3.0 was released in 1989 [5]. The interpreter was written in Fortran and C under the MS DOS operating system. FAT 3.0 allowed one to analyze and to optimize two-dimensional and axisymmetrical problems of electrostatic, magnetostatic and time harmonic electromagnetic fields.

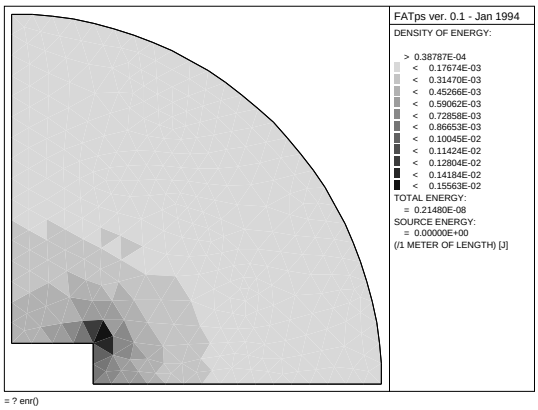

Fig. 1. User interface of FATpc 3.0 interpreter

FAT was primarily designed to be used for teaching. For many years it was successfully used for teaching of field theory as well as numerical methods for solving partial differential equations. The conception of FAT as the programming language, not GUI driven field simulator (as the most of such programs) is rather unusual. However, our experience has proven that this conception is excellent for teaching purposes. Easy and intuitive syntax allows a student to concentrate on the field model and simulation problem itself, not on the user interface to the simulator. Description of the boundary problem and simulation task as a computer program makes it human readable what allows analysis of the model for simple problems even without computer and makes the boundary problem portable.

Aside of the main application of FAT, several extended versions of the interpreter were made. They were used for our research work as well as for solving many practical problems. Many years of experience with FAT pushed authors to redesign the language and the interpreter to make it more expandable and portable. The new project, started in 2002 is aimed to design a completely new, objective, portable, public domain code of the FAT interpreter.

The objective design of the interpreter allows us to solve easily the most important problems of code portability and extensibility which were difficult to solve in procedural implementation of the previous version.

II. Object oriented programming

A. Software models

Every model of the program "life" can be divided into several stages. Stage is usually defined as the time between the "milestones" of the program production. Each stage ends with some goals reached and some documentation finished. The decision of the processing to the next stage is then taken by the project managers [3]. Many models of the program design were presented. Each of them emphasizes some aspects of the design process. One of the relatively new, is the spiral model, proposed by Boehm in 1988 [1] and shown in Fig. 2.

The spiral model presents development of the program as a continuous process of circulation between several stages. As the result of each circumference the new version of the program is released. Each repetition of every stage becomes longer, because the project grows and every time more work needs to be done. The spiral model assumes that iterations bring continuous, systematic growth of the system architecture and its possibilities. The iterative process can be seen to be driven by the

Authors are with the Institute of Theory of Electrical Engineering, Measurement and Information Systems, Warsaw University of Technology, ul. Koszykowa 75, 00-662Warsaw, Poland, email: *jstar@iem.pw.edu.pl*

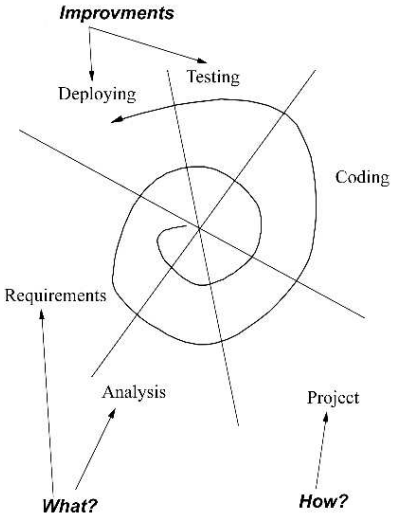

Fig. 2. Spiral model of the program life

risk—in each new version the most important dangers for success of the project should be eliminated or the project should be stopped if risk of fail becomes too large.

B. Object oriented model

Most of the new methodologies of software production are based on the object oriented point of view. In such approach the most basic construction block of the program is a class or an object. Object oriented programming sees programs as collections of objects which cooperate to solve the problem [3], [4]. Each object is an instance of some class and all classes build some kind of hierarchy tree, bounded together by superclass-subclass relations. Such hierarchy allows a program designer to place the common code in a super-class, and then design specialized code in sub-classes.

Each object represents an item from the problem domain. Class is a set of the similar objects, it can be also seen as the type of objects. Each object has its own identity (name), state (data) and set of operations which can be performed by this object or with this object. The other concepts which can and should be used in an objective code are polymorphism and late binding. Polymorphism means that a variable of a given class-type can represent not only an object of this class, but also an object of any of its sub-classes. Decision of the exact type of the object represented by a given variable is made, when an object is to be used—late binding means that the proper methods for the real object, not for the variable type are called [3].

III. Application design

A. Analysis of requirements

At that moment, the functionality of the application can be described with single scenario. It is typical for all field simulators. First of all, the input data must by read either directly from user's console or from another source (disk file). These data describe geometry of the boundary problem. Next, the defined domain should be divided with finite elements. After boundary conditions and source functions values are assigned properly, the main equations system can be formulated and solved. The final results must be presented to the user (what usually requires some post-processing of the data) and/or saved in the system resources.

Such a description of the simulator run assumes single and simple application control flow. Of course, it's not enough in a typical use that we expect, but such a short and simple description of the functionality allows us to identify most of the system's elements that resides on the highest abstraction layer. The abstraction shows some properties of class or the whole system and hides other ones, less important at the moment the analysis is made.

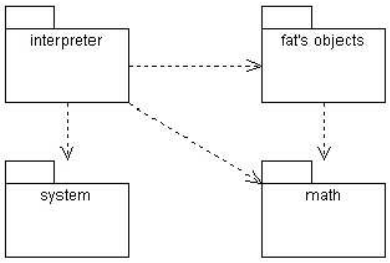

Fig. 3. Components of the FAT system

Based on the scenario presented above the four independent system's parts can be pointed (Fig. 3). The `system` module contains classes responsible for cooperation with underlying operating system, which includes data managing, graphical and I/O operations. The part called `interpreter` is used to check if the syntax of commands provided by user or read from file is correct. The second task of the `interpreter` is to pass the control to the appropriate program's part. All classes used to hold information about the boundary problem are grouped in the `fat's objects` module. The last part—`math`—keeps definitions of the mathematical expressions' syntax. It also contains all routines necessary to manage those expression. All associations between program's parts are shown in Fig. 3.

B. Class identification

The object model of the real world is built in the object oriented analysis. During that process the class's requirements are investigated. The work results are used theh as the base for object oriented developing. At the very beginning, some classes that describes problem need to be known. It helps to define bounds within which the further exploration will be made. When the first classes are found, we can say that abstraction is formulated (highest possible). In the next iterations, lower and

more detailed layers are built. When that process is finished, the collection of those abstractions aims the solution of the undertaken problem.

There are many different analysis methods. The simplest possible was chosen for our use. It's called "linguistic based information analysis". In order to perform such an analysis, good, formal, human readable description of the problem is needed. In the given description all verbs and nouns must be marked. The nouns indicate the potential classes. The verbs shows operations that class will have to do.

In the case of FAT system, the brief characteristic of the finite element method and idea of 2D geometry description is necessary starting point for the objective analysis [2]. The main idea of the FEM method is the division of the investigated domain (containing one or more sub-domains which we call macroelements) into number of small elements of simple shape. Each element is contains nodes in which values of some field quantity should be calculated (we do not consider edge-elements). The domain is approximated with straight line segments as the elements' bounds are the straight line too (linear elements). The boundary condition can be assigned to any segment or node. The source function values and materials can be defined differently for any macroelement.

Based the description presented above, the following classes were identified: *macroelement*, *element*, *node*, *boundary condition*, *material* and *macroelement's boundary*. Some new classes can be found, if we step down into the abstraction layers hierarchy. The bound of the *macroelement* is built with sequences of the *segments* (we shall call it a *chain*). Each segment has its own beginning and end—those points will be called *basis points*. Any geometrical curve can be used to build the *segment*, for example: *line*, *circle*, *spline*. The *materials* can be linear and nonlinear.

As can be reviewed, the collection of the classes, that are unconnected with any association rules, was produced as the effect of the object oriented analysis.

C. Design of main classes

The responsibilities of the particular objects must be defined now. As the responsibility we understand tasks including attributes and methods, the objects should have, use and share [3], [4]. It's the right time to find the attributes and methods which the known classes are made of. The case analysis was used to find out the methods that each object must have. The case is the action taken by a user that forces system to work in the specific way. Using the case analysis, we know for example that a basis point ought to generate (`mkNds()`) and delete (`rmNds()`) a node. When the system requires that, any object should present its name (`objName()`) and type (`objType()`). Additionally, there are defined methods used to print, draw and manage data within the object.

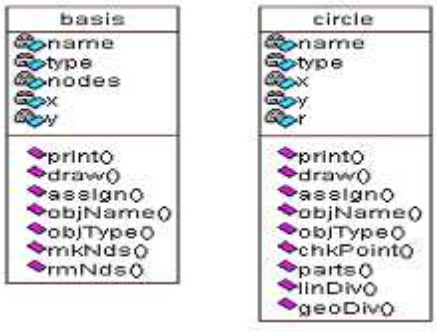

Fig. 4. Specification of two example classes

The complete specification of the basis point class is present in Fig. 4. It contains all other association discovered during the developing process. This specification contains the necessary methods (functions) and atributes (fields). These attributes are needed for object to operate properly. The `x` and `y` are the coordinates of the point in 2D geometry. Atribute `name` identifies the object and `type` helps to manage the data structures and makes the program works smoothly. The `nodes` attribute points to node that was generated by the basis point.

On the right side of Fig. 4, the declaration of class `circle` is shown. All methods and attributes were discovered exactly the same way. It's worth to notice that despite objects are used to hold totally different data some methods and attributes are common. What's more, nodes attributes and `mkNds()`, `rmNds()` methods are common for small group of objects that contain: *basis point*, *chain* and *segment* classes.

These facts will be used to refine object model in the next stage of developing process.

D. Relations identification

Now, the association between classes must be discovered and classified. In order to find complex associations rules the incremental method must be used. It means that associations rules must envelop while the system grows. In fact a lot of discovered association will explore collaboration paths between abstraction layers.

It was noticed earlier that each object has methods called `objName()` and `objType()` used for an identification by the system. Another common part of all classes are interfaces for printing, drawing and data managing. That's why a new class called `fatobj` was discovered. Such an object will never be created during the program runs—it's only used to group and share all methods common for all objects that inherit from it. There are much more generalizations that can be uncovered in the FAT

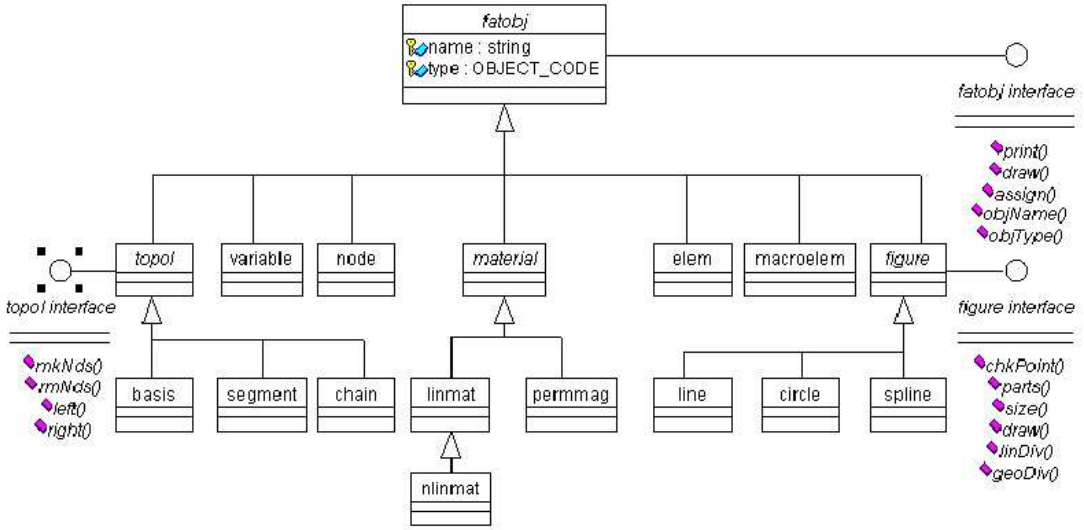

Fig. 5. Class hierarhy for FAT objects

system. Most of it concerns only a small part of all classes, but these relationships can improve the object scheme and makes the final program code less complicated and far more effective. In Fig. 5 the complete inheritance hierarchy is presented.

The `fatobj` is the root of the hierarchy tree. As an abstract class it assure that all classes below it will share part of the interface. It's absolutely necessary, if the similar behavior of the whole system, independently on the situation must be kept. All objects used to describe the boundaries of the simulated domain are derived from the `topol` abstract class. Similarly, `figure` class is the highest leaf of the sub-tree that contains objects representing geometrical curves.

IV. Implementation

The C++ was chosen as the production language [6]. Creation and managing of the new classes is very fast and easy in C++ which fully supports the object oriented programming. The complicated hierarchies of the objects can be built including single and multi inheritance. Polymorphism and late binding mechanisms are also found in the C++ what allows to differ object behavior while the program runs. The C++ language is known to be most effective and popular. There are many routines, that are made in C++ by other authors, which can be attached to the application under developing.

During the implementation all classes must be coded in the production language. It allows programmer to improve the project, so that new classes could be eventually found on lower abstraction layer. When the basis implementation exists the current iteration of developing process can be considered to be done. The whole project will envelope further in the next iterations.

V. Conclusions

Due the limited space only the first (but most important) phase of the objective FAT interpreter was presented. Current version of the project lacks some functionality of the older version, but the most important kernel of the simulator is already finished. This kernel analyzes of the FAT language program and stores all data necessary to define a boundary problem in sort of "abstract model". The discrete, numerical model is created only if a user requests mesh generation or filed simulation. Such approach allows one to make the model parametric—dimensions,materials, boundary conditions, etc. can be defined as functions of several parameters what can be used for optimization or variant calculations.

The kernel of interpreter can work with different field simulators. FAT is designed to be an user front-end to the filed simulator. Currently the old FEM engine, designed for the previous version of FAT can be used, but a new objective engine based on public domain Getfem++ package [7] is being tested.

References

[1] B. Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, Vol. 21, No. 5, pp. 61-72, 1988.
[2] Bolkowski S. et. all: Komputerowe metody analizy pola elektromagnetycznego. Warszawa: WNT 1993.
[3] Booch G.: Object-Oriented Analysis and Design with Applications. California: The Benjamin Cummings Publishing Company, Inc. 1994, second edition.
[4] Brumbaugh D.: Object-Oriented Development Building CASE Tools with C++. New York: John Wiley & Sons, Inc. 1994.
[5] FAT Reference Manual (for FATpcv, ver. 3.2 and above). Warszawa: IETIME Politechniki Warszawskiej 1993. `http://www.iem.pw.edu.pl/zetis/oprogramowanie/FAT/`
[6] S. Lippman, Inside the C++ Object Model, Addison-Wesley Longman , Reading,MA, 1996.
[7] `http://www.gmm.insa-tlse.fr/getfem/`