OBJECT-ORIENTED GRAPHIC ENVIRONMENT FOR STRUCTURAL ANALYSIS

R. Robert Gajewski[#], Tomasz Kowalczyk[##]

[#] Warsaw University of Technology, Center of Computer Methods
Aleja Armii Ludowej 16, 00-637 Warszawa, Poland
E-mail: rg@omk.il.pw.edu.pl
on leave at: University of Essen, Department of Civil Engineering
Universitatstrasse 15, D-45117 Essen, Germany
E-mail: rg@bauwesen.uni-essen.de

[##] The University of Tokyo, Faculty of Engineering
Department of Quantum Engineering and System Science
7-3-1 Hongo, Bunkyo-ku, 113 Tokyo, Japan
E-mail: kowal@thyme.gen.u-tokyo.ac.jp

Extended abstract

In the present days finite element method (FEM) seems to be established as the most efficient and popular tool used to solve complex problems arising in computational mechanics. As the result of more than three decades of development there are many robust FEM codes written usually in FORTRAN. The usage of graphical pre- and post-processors is getting more interest but these modules are rather not well integrated with the solution part. All huge FEM codes written using procedural programming paradigm are difficult to understand, maintain and expand. On the other hand one of the main features of the life-cycle of a FEM program is its evolutivity, mainly due to the fact that only few details can be fully foreseen at the time of the initial design of a program. It results in the situation where the most efforts invested in such program are spent mostly during its evolution, whereas the share of preliminary phase is usually small. Apart from such challenges for the developer of FEM code like robustness, reliability, optimised speed and memory requirements this code should be, from the users point of view, user friendly and as intuitive as possible. All users need working environment giving them precise understanding of the solved problem and moreover possibilities of the interaction with the solution process.

Increasing rapidly in the last few years capabilities of the object-oriented (OO) techniques made them a very promising tool for the development of large-scale codes. Since OO methods encapsulate data and methods codes can be written in the term of underlying physics of the problem with less regard for computer details. Moreover all objects could have graphical input as well as viewing methods attached to them in the same way as pure mathematical methods. This facility has great advantages in the terms of library development, since graphical methods can be attached independently from mathematical, enabling incremental development of the code and straightforward modifications.

Graphic User Interfaces (GUIs) are becoming standards for user-oriented applications. They are consistent across applications and easy to learn and use. Each finite element program invariably consists of a pre-processor, processor and post processor, which are in the majority of

commercially available packages still three separate blocks of code. Pre- and postporcessing modules have become an important part of any finite element system. In future the analysis modules will rather be part of graphical modules and not the other way round. Therefore it is imperative to employ novel software structures if these objectives are to be met. While designing an application it is reasonable to separate the part of the system responsible for calculations, i.e. the solver, from the GUI classes to make them independent and thus more flexible. By doing so we encounter problems, how to assume relations between these modules.

The essence of GUI are *GraphicObject*s. These objects posses two important abilities. Firstly, they can display themselves on the specialised *ProjectionWindow*, and next, they can communicate with the user (e.g. through the dialogue box), to let him change their characteristic features. It is obvious that some of the *GraphicObject*s should represent *Node*s, *Element*s and *Load*s which are the part of the solver and so, as the objects responsible for calculations should not make assumptions about the graphics.

There are generally two ways of designing *GraphicObject*s. In the first full independence of the solver from the graphic environment is assumed. The solver and its classes, written in the purely portable code, are completely separated from the rest of the system. *GraphicObject*s are not derived from the objects they represent. They are only *Handle*s to them. Additional messages between the modules are necessary - from the *Handle*s to the objects they represent, in order to retrieve their data (see Figure 1).

In the second solution the solver and the graphic environment are connected through the same objects they operate on. *Node*s, *Element*s and *Load*s are themselves *GraphicObject*s through derivation. Thus both *Solver* and the graphic environment can operate directly and independently on the same objects. It is important that we can still logically separate the two parts of the classes: responsible for calculations and responsible for representation. No additional messages between the two modules are here necessary.

Due to the lack of fully portable graphic library first solution was chosen.

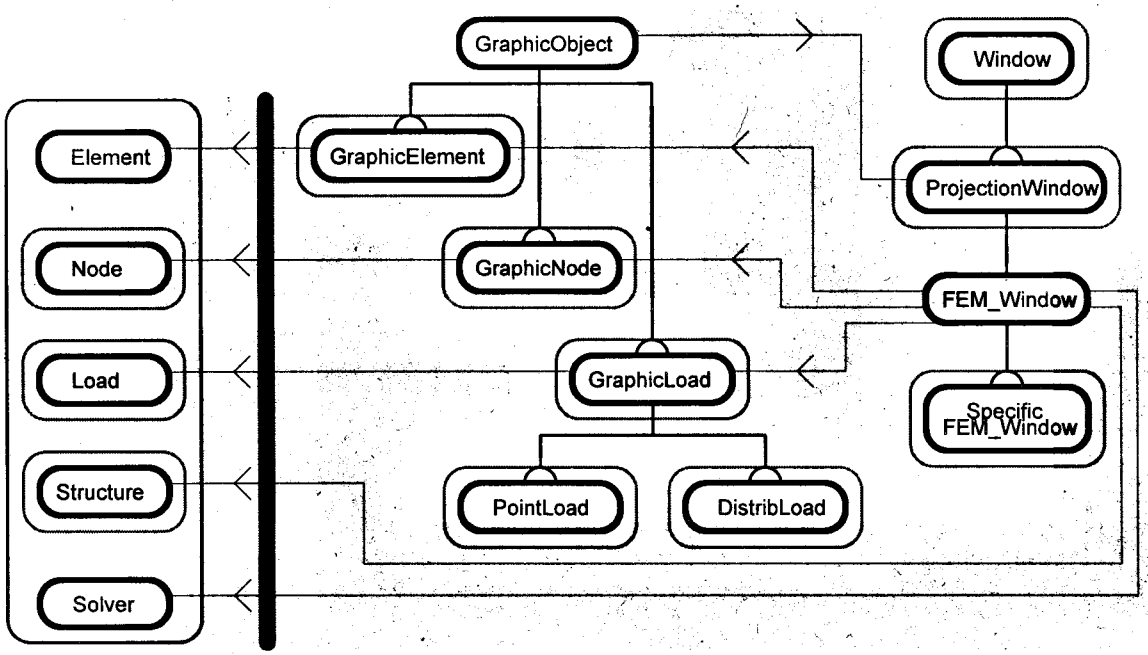

Figure 1. Class hierarchy of an object-oriented graphic environment.