Design Patterns for Object-Oriented Scientific Software

Serhii Choporov[1], Serhii Gomenyuk[1], Oleksii Kudin[1], and Andrii Lisnyak[1]

[1] Zaporizhzhya National University, Zaporizhzhya, Ukraine
s.choporoff@znu.edu.ua, gserega71@gmail.com,
alexkudin@znu.edu.ua, andrey.lisnyak@gmail.com

Abstract. Software design patterns are general reusable object-oriented solution. In software engineering, patterns have been proven to offer many benefits. Scientific software also becomes more object-oriented and the importance of design patterns increases. We present a set of design patterns for object-oriented scientific software. Particularly we develop computer-aided engineering software based on the Finite element method. Initially, we decompose the problem into subsystems by applying the commonality and variability analysis. A set of commonalities includes following terms: a representation scheme, a mesh, a solver, etc. A representation scheme is an interface that allows to check whether a point belong to a solid or not. A mesh is the discrete representation of the solid via a set of simple geometric shapes. Four basic design patterns for the scientific software development have been presented in this paper. There are developed UI–Model–Analysis, Representation–Mesh, Element–Mesh, and FEA Problem patterns. These design patterns separate pre-processing, the analysis solver, and post-processing of results.

Keywords: Software Engineering, Software Design Patterns, Scientific Software, Finite Element Method, Object-oriented Approach.

1 Introduction

Software design patterns as general reusable solutions were introduced in the end of the 1980s and, since that time, they have been actively explored in software engineering [1-3]. Until recently, scientific programmers have usually avoided object-oriented approaches because of their heavy computational over-head [4]. However, scientific software becomes larger and requires flexibility, extensibility and maintainability [5]. Design patterns deal with these issues providing generic object-oriented solutions.

CAE software is a kind of scientific software that areas may include the stress analysis, the thermal analysis, the fluid flow analysis, the multibody dynamics etc. In general, a CAE system consists of three subsystems: pre-processing, an analysis solver, and post-processing of results.

2 Catalogue of Design Patterns

Consider a generic CAE system that uses the finite element method in the solver subsystem. We can assume that every solid's model is initially defined by some representation scheme and then this model is discretized into a mesh.

2.1 UI–Model–Analysis Pattern

The most modern CAE software have integrated graphical user interface (UI). Using UI controls, user defines the model of the problem and performs the analysis. Hence, three main packages participate in the high-level decomposition.

The UI package contains classes that implement UI-related responsibilities of software. The analysis package is responsible for the numerical analysis. The model package defines the interface that allows to check whether a point belong to a solid or not. Naturally, there should be no coupling between the analysis and UI subsystems. However, the UI package is dependent on pre-processing, the solver and post-processing.

2.2 Representation-Mesh Pattern

We suppose that a domain model is described in terms of some representation scheme. Boundary representation (BRep), constructive solid geometry (CSG), and function representation (FRep) are the most commonly used schemes. A common property of CSG and FRep that it is easy to check whether arbitrary point belong to the solid or not. We also can assume that a mesh is an abstract interface that allows generating and iterating over a collection of elements. In this case, concrete classes derived from the Mesh class generate collections of elements with appropriate shape using an abstract representation' interface to classify a point. Both Representation and Mesh classes participate in the Representation–Mesh pattern (see Fig. 2). The intent of this pattern is to separate responsibilities between representations and mesh generation classes.

Both Representation and Mesh classes are an application of the Strategy pattern [1]. The Mesh class and its derivatives can also be implemented as Iterator [1] to traverse a collection of elements. In addition, some meshing algorithms may be implemented as Template Methods [1].

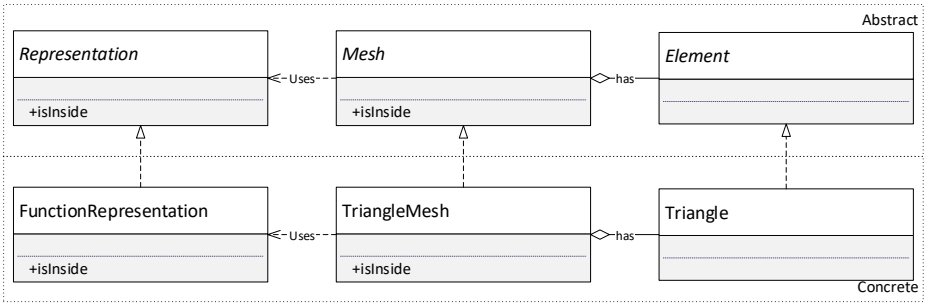

Fig. 1. The Representation–Mesh pattern

2.3 Element–Node Pattern

In general, an element is a collection of nodes in order is significant. Both two- and three-dimensional elements have edges (an edge is a straight-line segment connecting two nodes). In addition, three-dimensional elements have faces that are flat elements enclosed by edges. Thus, the concrete face is an object of the class that inherits the Element interface. Hence, Element, Face, Edge, and Node are structural elements of the Element–Node pattern (see Fig. 2).

The Element class and its derivatives (including faces) can be implemented as Iterator [1] to traverse collections of points and edges. The Iterator pattern can also be employed in the Node class and its derivatives to iterate a set of adjacent elements.

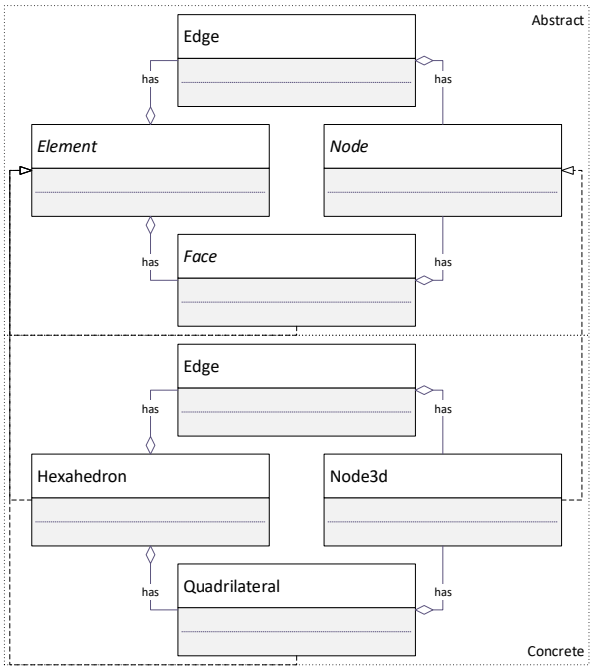

Fig. 2. The Element–Node pattern

2.4 FEA Problem Pattern

The FEA Problem pattern can be derived from the Template Method Pattern [1] adding mesh, boundary conditions, and forces (see Fig. 3). The intent of this pattern is to define a skeleton of a FEA algorithm and the object composition for boundary conditions and forces, which participate in the problem.

Forces and boundary conditions implement the FeaValue interface. This interface allows to obtain the direction and the value in any point. Forces and boundary conditions are usually specified by the UI or DSL model. However, using the Adapter pattern [1], we can implement the FeaValue interface.

3 Conclusion

This paper has been proposed an approach for the development of scientific software using design patterns. Particularly, four basic design patterns for the finite element programming have been presented in this paper. The first, the UI–Model–Analysis pattern decomposes software into high-level subsystems. The second, the Representation–Mesh pattern separates relations between representations and mesh generation classes. Next, the Element–Node pattern uses object decomposition to define elements of a mesh. Last, the FEA Problem pattern defines the structure for a generic finite element problem. These patterns show how object can be organized for greater flexibility and maintainability. Patterns represent abstractions of the CAE design without restrictions on the source code.

Acknowledgments. This research is funded by The Ministry of Education and Science of Ukraine.

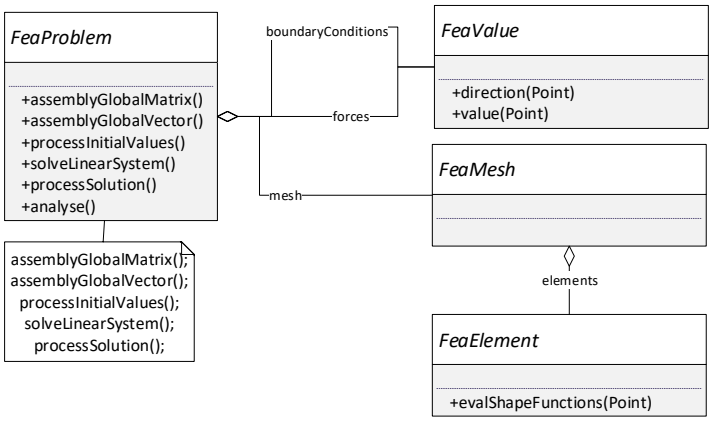

Fig. 3. FEA Problem Pattern

References

1. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1994).
2. Alexandrescu, A.: Modern C++ Design: Generic Programming and Design Patterns Applied. Addison-Wesley (2001).
3. Shalloway, A., Trott, J.R.: Design Patterns Explained: A New Perspective on Object-Oriented Design. Addison-Wesley (2004).
4. Blilie, C.: Patterns in Scientific Software: an Introduction. Computing in Science and Engineering 4(3), 48-53 (2002).
5. Cickovski, Tr., Matthey, Th., Izaguirre, J.A.: Design Patterns for Generic Object-Oriented Scientific Software. Technical Report TR 2004-29. University of Notre Dame, Notre Dame (2004).
6. Heng, B.C.P., Mackie, R.I.: Using design patterns in object-oriented finite element programming. Computers and Structures 87(15–16), 952–961 (2009).