Chapter VI

Automatic Verification of OOD Pattern Applications

Andrés Flores, University of Comahue, Argentina
Alejandra Cechich, University of Comahue, Argentina
Rodrigo Ruiz, University of Comahue, Argentina

Abstract

Object-oriented patterns condense experimental knowledge from developers. Their pragmatic benefits may involve a reduction on the effort impact of the maintenance stage. However, some common problems can be distinguished as well. For instance, some design patterns are simply too difficult for the average OO designer to learn. A pattern-based design process could be enhanced by the provision of an automatic support for modeling and verification with a proper formal foundation. In this chapter we show how formal specifications of GoF patterns have been helpful to develop that tool support, where we have adopted the well-known Java language upon its portability facet. Thus, we are changing the object-oriented design process.
by the inclusion of pattern-based modeling and verification steps. The latter involving checking design correctness and appropriate pattern application through the use of the supporting tool, called DePMoVe (Design and Pattern Modeling and Verification).

Introduction

Object-oriented patterns represent an important source of knowledge by condensing years of experience from developers. They certainly became a useful technical vocabulary which helps for developers to discuss and communicate ideas. Particular design problems, for example, can be easily described by “using an observer” or “separated by a bridge.” The abstractness of patterns description allows cutting across traditional boundaries of system development and business engineering. At any stage of a development project, knowledge upon business processes can be practically shared when patterns are properly applied on the supporting systems.

The pragmatic benefits of using design patterns may involve the possibility to reduce the impact on cost — time + effort — of the maintenance stage, which usually may rise to 80% of the overall cost of the project (Polo, Piattini, & Ruiz, 2002). Design patterns address the concern of the evolutionary nature of the software, since they may allow software to be flexibly accommodated to constant changes. Hence, they are considered as a technique for design to change, thus satisfying the design principle of anticipation to change (Gamma, Helm, Johnson, & Vlissides, 1995; Ghezzi, Jazayeri, & Mandrioli, 2002).

However, a number of common problems can be distinguished as well. For example, some design patterns are simply too difficult for the average OO designer to learn. In some cases, the pattern is inherently complex; in other cases, the pattern involves an explanation and a name that are not obvious. Hence, pattern understanding could be a serious problem for being skeptic when they will be adopted by an organization. In practice, an OO designer needs personal time and personal initiative to become skillful in design patterns. This could be approached by firstly experimenting with the use of a pattern in a toy example before it can be used in the real case. Thus, the main issue implies OO designers with an imperative need of learning how to apply design patterns.

Such a misunderstanding problem on patterns applications has been distinguished as a consequence of the provided description on current patterns catalogues (Alagar, & Lämmel, 2002; Eden, 2000; Lauder, & Kent, 1998). Patterns are invariably described informally in the literature (Buschmann, Meunier, Rohnert, Sommerland, & Stal, 1996; Cooper, 2000; Coplien, 1996; Fowler, 1997; Gamma,