Chapter XV

A Pattern Enforcing Compiler (PEC) for Java: A Practical Way to Formally Specify Patterns

Howard Lovatt, Macquarie University, Australia

Anthony M. Sloane, Macquarie University, Australia

Dominic R. Verity, Macquarie University, Australia

Abstract

This chapter describes an extended compiler that formalizes patterns, which we call a pattern enforcing compiler (PEC). Developers use standard Java syntax to mark their classes as implementations of particular Design patterns. The compiler is then able to use reflection to check whether the classes do in fact adhere to the constraints of the patterns. The checking possible with our compiler starts with the obvious static adherence to constraints such as method presence, visibility, and naming. However, we go much further and support dynamic testing to check the runtime behavior of classes and code generation to assist in the implementation of complex patterns. The chapter gives examples of using the patterns supplied with our PEC and also examples of how to write your own patterns and have our PEC enforce these.
Introduction

Formalization techniques for Design patterns vary in the extent to which mathematical formalisms are employed. While it is possible to model many aspects of patterns in a very precise way with mathematical techniques, the result is often methods that have restricted scope to deal with systems on a real-world scale or tools that are unlikely to be integrated into developers work practices due to unfamiliarity and training requirements. This observation has been made a number of times in the more general area of formal methods for software development.

This chapter is based on our research in using compiler technology to automatically enforce Design patterns in Java code. This approach provides many of the benefits of more mathematical approaches, such as mechanized checking, but also enables us to integrate the checking with standard developer practices. For example, our approach supports standard compilation and build tools as used by modern Integrated Development Environments.

Our approach achieves pattern formalization via an extended compiler, which we call a pattern enforcing compiler (PEC). Developers use standard Java syntax to mark their classes as implementations of particular Design patterns. The compiler is then able to use reflection to check whether the classes do in fact adhere to the constraints of the patterns. The checking possible with our compiler starts with the obvious static adherence to constraints such as method presence, visibility, and naming. However, we go much further to support dynamic testing to check the runtime behavior of classes and code generation to assist in the implementation of complex patterns. We have a robust implementation, structured as a transparent back-end to any standard Java compiler, and it is freely available by download (http://pec.dev.java.net) under the Lesser GNU General Public License (Lovatt, 2004). In the standard distribution we support a number of standard Design patterns, but developers can easily extend the system.

The chapter is structured as follows:

- Review the literature on related work to give background to the work and to tease out problems with current approaches.
- Introduce the idea of a PEC and outline the philosophy and motivation behind our approach to pattern formalization to show why this approach solves many of the problems highlighted when reviewing literature.
- Give some examples of simple patterns to illustrate how developers can formalize their own patterns.
- Introduce a more complex pattern that is derived from a formal specification.
- Briefly describe the patterns that our PEC currently supports.
- Suggest future trends.
- Conclude and summarize the chapter.

The main thrust of this chapter is to demonstrate how a contract for a Design pattern can be enforced using our compiler; but before getting into details it is useful to provide an example of writing a class that is enforced, so that context is provided for subsequent discussions.
Rotation Invariant Texture Image Retrieval with Orthogonal Polynomials Model
www.igi-global.com/article/rotation-invariant-texture-image-retrieval/64184?camid=4v1a

A Blind 3D Watermarking Approach for 3D Mesh Using Clustering Based Methods
www.igi-global.com/article/a-blind-3d-watermarking-approach-for-3d-mesh-using-clustering-based-methods/87250?camid=4v1a