Chapter X

Modeling and Reasoning about Design Patterns in SLAM-Sl

Angel Herranz, Universidad Politécnica de Madrid, Spain

Juan José Moreno-Navarro, IMDEA Software, Spain

Abstract

In this chapter, a formal model for Design patterns is studied. The formal specification of a Design pattern is given as a class operator that transforms a design given as a set of classes into a new design that takes into account the description and properties of the Design pattern. The operator is specified in the SLAM-Sl specification language, in terms of pre and postconditions. Precondition collects properties required to apply the pattern and postcondition relates result classes encompassing most of the intent and consequences sections of the pattern. Formalization is mandatory for reasoning about Design patterns and for implementing assistant tools.
Design patterns (Gamma, Helm, Johnson, & Vlissides, 1995) and refactoring (Fowler, 1999) are two sides of the same coin. The aim of the application of both concepts is creating software with an underlying high quality architecture. Design patterns are “descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context” (see Gamma et al., 1995). Theoretically, the domain of application of Design patterns is the set of problems. Refactoring is “the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure” (Fowler, 1999). Its application domain is the set of solutions. In practice, Design patterns are not directly applied to problems but to incomplete solutions, to conceptual solutions, or to concrete solutions. In many cases Design patterns are model refactoring descriptions.

According to Tokuda (1999) and Cinnéide (2001), automating the application of Design patterns to an existing program in a behavior preserving way is feasible. In other words, refactoring processes can be automatically guided by Design patterns. In this work, we describe our pattern design formalization technique and its relation with design and refactoring automation (as well as other useful applications).

The first unavoidable step is to introduce a formal specification of Design patterns. This will be done in terms of class operators. Then, some practical applications will be presented, including how to use the formalism to reason about Design patterns and how to incorporate this model into design tools and software development environments.

Let us provide an informal and intuitive description of our proposal. A given (preliminary) design is the input of a Design pattern. This design is modeled as a collection of classes. The result of the operation is another design obtained by modifying the input classes or by creating new ones, taking into account the description of the Design pattern.

For instance, consider you have an interface Target that could be implemented by an existing class Adaptee but its interface does not match the target one. The Design pattern Adapter, considered as an operator, accepts classes Target and Adaptee as input, and returns a new class Adapter that allows for connecting common functionality. Similarly, when a client needs different variants of an algorithm, it is possible to put each variant of the method in different classes and abstract them, automatically “introducing” the abstract class that configures the Strategy pattern.

A function that models the Design pattern is specified in terms of pre and postconditions. A precondition collects the logical conditions required to apply the function with success. In our case, it allows specifying some aspects of the Design pattern description in an unambiguous way. Talking in terms of the sections used to describe a pattern, the pattern function precondition establishes the applicability of the pattern. For instance, in the pattern Strategy, the precondition needs to ensure that all the input classes (concrete strategy) define a method with the same signature. A postcondition relates input arguments and the result. In the Adapter function, the postcondition establishes that input classes (Target and Adaptee) are not modified, and that a new class (Adapter) is introduced, inheriting from the input classes. The Adapter’s methods are described by adequate calls to the corresponding Adaptee methods. The postcondition encompasses most of the elements of the intent and consequences sections of the pattern description.
Related Content

Evaluation Approach of Arabic Character Recognition
www.igi-global.com/article/evaluation-approach-arabic-character-recognition/55100?camid=4v1a

A Marked Point Process Model Including Strong Prior Shape Information Applied to Multiple Object Extraction From Images
www.igi-global.com/article/marked-point-process-model-including/55096?camid=4v1a

A Highly Efficient Content Based Approach to Filter Pornography Websites
www.igi-global.com/article/highly-efficient-content-based-approach/68005?camid=4v1a
Image Matting Based Multi-Focus Image Fusion With Optimal Cluster Size
[www.igi-global.com/article/image-matting-based-multi-focus-image-fusion-with-optimal-cluster-size/212375?camid=4v1a](www.igi-global.com/article/image-matting-based-multi-focus-image-fusion-with-optimal-cluster-size/212375?camid=4v1a)