Chapter 7

Fine Grained Patterns for Real-Time Systems

Bruce Powel Douglass, Ph.D.
I-Logix

Abstract: A design pattern is a generalized approach or solution to a commonly occurring problem. Design patterns are a way of capturing and codifying design expertise in the forms of solutions that have proven effective in solving specific kinds of problems in a various contexts. This chapter discusses how fine-grained patterns that solve specific kinds of problems common in real-time systems. These and other related patterns can be found in the references. Much of this material is adapted from the author’s book Real-Time Design Patterns: Robust Architecture for Real-Time Systems (Addison-Wesley, 2002).

Key words: Real-time design patterns, resource management patterns

1. INTRODUCTION

Collaborations define the structural elements and their relations necessary to achieve a use case behavior. Patterns exist to provide common ways of wiring together collaborations that optimize some criteria of importance. Because the functional issues are addressed by the collaboration itself, design patterns optimize some quality of service (QoS) criteria such as performance, memory usage, reusability, robustness, safety, or reliability. Rather than exhaustively catalog all potentially useful patterns, let us concern ourselves with patterns that address some specific problems faced by real-time and embedded system developers. Catalogs of patterns useful for real-time systems are dealt with in more detail elsewhere [2, 3, 4] and the interested reader is referred there for a more complete set of real-time design patterns.

Experienced developers find when they approach a new problem, that the situation usually has something in common with a solution they have previously either created or seen. The problems are not identical and the
identical solution will rarely solve the new problems, but the problems are nevertheless similar, so a similar solution will probably work. The “similar solution,” generalized and formalized, is called a design pattern. Creating design patterns is a problem of abstracting the similarities of the many specific instances of the problem and their solutions. The discovered generalized solution can then be applied to the new problem at hand.

Of the three fundamental concerns associated with patterns, the first has to do with the application of patterns. The problem of identifying the nature of the problem and examining the patterns “library” for the best ones to apply is called pattern hatching[10]. And, as John Vlissides, author of that excellent book points out, this name implies that we’re not creating something new but “developing from preexisting rudiments.” These preexisting rudiments are our captured design patterns that we can use to construct solutions that work in novel circumstances.

Another issue, of course, is the identification and capture of new patterns to add to the library. This process I call pattern mining. It involves the abstraction of the problem to its essential properties, creating a generic solution and then understanding the consequences of that solution in the problem context in which the pattern applies.

Lastly, patterns must be instantiated – that is, they must be applied to the application problem at hand. This is usually a combination of specialization of the general pattern roles, parametric instantiation of parameterized classes, and modification of application classes to take on properties of the pattern elements.

Patterns are not just software reuse, but rather a kind of concept reuse. Most patterns, such as those presented in this book are related to design. Design is always an optimization of an analysis model and design patterns are always a general concept for how to optimize an analysis model in a particular way with particular effects.

Optimization always entails improving some aspect of the system at the expense of others. For example, some patterns will optimize reusability at the expense of worst-case performance. Other patterns will optimize safety at the expense of system recurring cost (i.e., cost per shipped item). Whenever you optimize one set of aspects, you necessarily deoptimize others. This is a fact of life, else we would all be driving at the speed of sound using no gasoline and in perfect safety for zero cost.

1.1 What is a Design Pattern?

A design pattern is “a generalized solution to a commonly occurring problem.” To be a pattern, the problem must recur often enough to be usefully generalizable. The solution must also be general enough to applied