Chapter IV

Patterns of Collective Behavior in Ocsid

Joni Helin, Tampere University of Technology, Finland

Pertti Kellomäki, Tampere University of Technology, Finland

Tommi Mikkonen, Tampere University of Technology, Finland

Abstract

This chapter presents an abstraction mechanism for collective behavior in reactive distributed systems. The mechanism allows the expression of recurring patterns of object interactions in a parametric form, and the formal verification of temporal safety properties induced by applications of the patterns. The abstraction mechanism is defined and compared to Design patterns, an established software engineering concept. While there are some obvious similarities, because the common theme is abstraction of object interactions, there are important differences as well. The chapter discusses how the emphasis on full formality affects what can be expressed and achieved in terms of patterns of object interactions. The approach is illustrated with the Observer and Memento patterns.
Introduction

The motivation for Design patterns is to improve the quality of software by improving its structure. Taking this one step further, the motivation for formalizing Design patterns is to improve their quality. Here, the premise of formalization is twofold. First, the commonly acknowledged benefits stemming from rigorous specification are the decrease of ambiguities and the flagging of hidden assumptions. In the case of patterns, this reduces problems that arise from their inappropriate application due to misunderstandings. Second, the possibility to mechanically reason about important properties of patterns contributes to better quality by finding errors and inconsistencies.

The interests driving this chapter are somewhat orthogonal to the issues addressed by the well known object-oriented Design patterns. Since Design patterns have their roots in the object-oriented programming community, it is natural that many patterns deal with object-oriented concepts. There are patterns for organizing inheritance hierarchies, traversal of structures composed of objects, and so forth. Many of these patterns are concerned with static organization of code rather than its behavioral properties, which are the focus of our approach.

The fundamental goal behind this chapter’s approach is the modularization of distributed behavior, which involves coordinated cooperation of objects over time. The units of behavior to be modularized are usually below the level of complete distributed algorithms. They are pieces of behavior that ensure that the system has some meaningful temporal properties. The background for the work of this chapter is in stepwise refinement, where the temporal properties of interest are often invariants that link abstractions with their implementations. Another concern that comes up in the context of distribution is atomicity.

Focusing on behavioral properties leads to somewhat different territory than the mainstream Design pattern work. A popular figure of speech when talking about the quality of code is “code smell,” which is a somewhat subjective, non-quantifiable property. Design patterns help to remove these smells by qualitatively improving the code. Temporal properties, on the other hand, are very clear cut. For example, an interaction either is atomic or it is not, and there are no intermediate degrees of atomicity.

Patterns are often described using UML (Rumbaugh, Jacobson, & Booch, 1998). For the purposes of this chapter, the problem with the UML mechanisms for describing behavior (i.e., sequence diagrams and collaboration/communication diagrams) is that, at least at the present time, they do not specify atomicity except at a very detailed level. This means that one cannot reliably reason about what happens when concurrent activities described using abstract diagrams interact, which defeats the goal of this chapter.

This chapter develops an approach based on views, which allow to modularize specifications based on their behavioral properties. Classes of similar views can be abstracted using view templates, which are parametric descriptions of collective behavior. Verification effort can be reused by verifying behavioral properties at the template level. The approach is embodied in the experimental Ocsid language (Helin & Kellomäki, 2004, 2005).

To avoid confusion, patterns of collective behavior are referred to as Ocsid patterns in the following, and the term Design pattern refers to patterns as understood by the object-oriented community.
Feature Extraction Methods for Intrusion Detection Systems
[www.igi-global.com/chapter/feature-extraction-methods-intrusion-detection/77588?camid=4v1a](www.igi-global.com/chapter/feature-extraction-methods-intrusion-detection/77588?camid=4v1a)

New Redundant Manipulator Robot with Six Degrees of Freedom Controlled with Visual Feedback