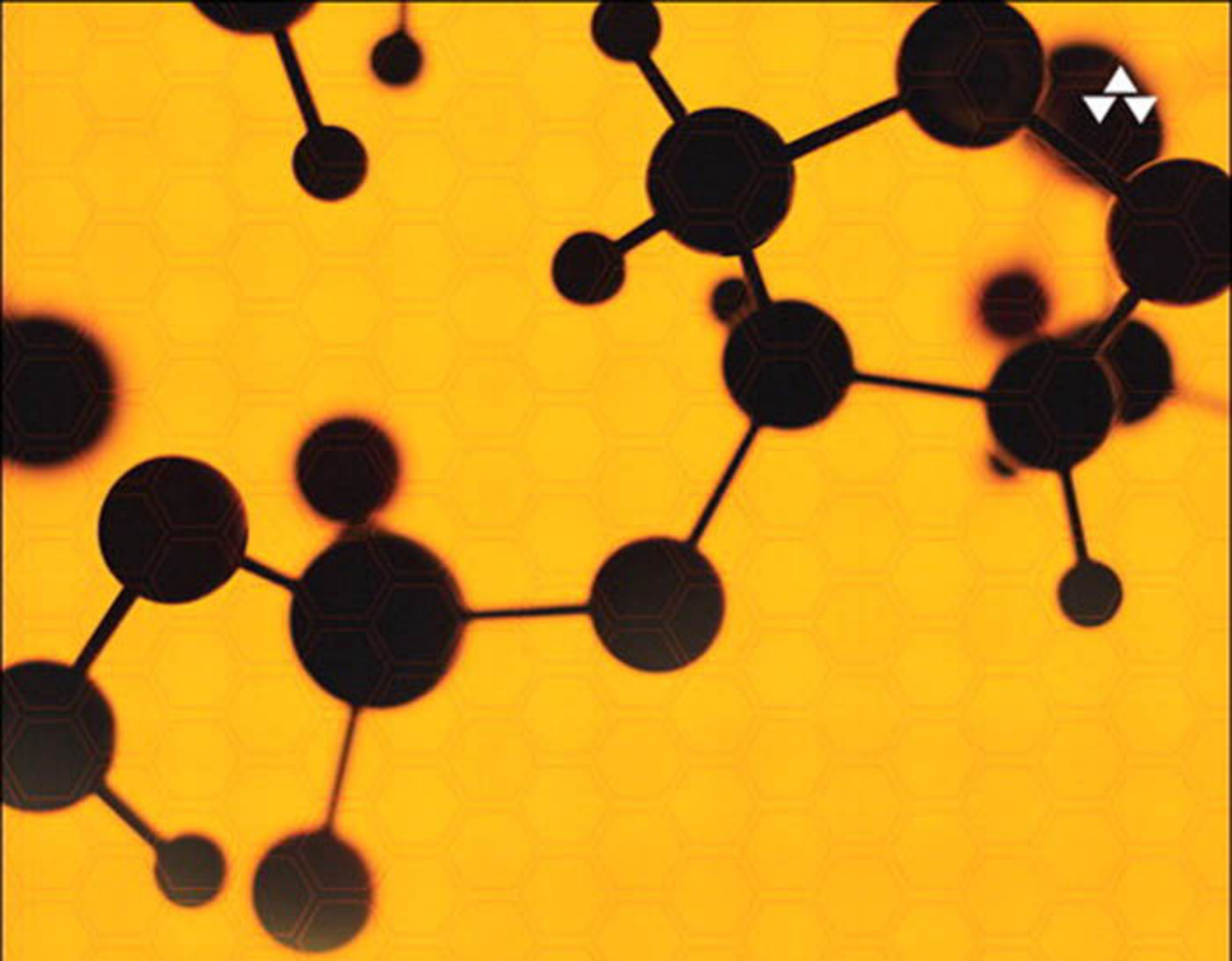

Elemental Design Patterns

JASON McC. SMITH

Foreword by **Grady Booch**

Elemental Design Patterns

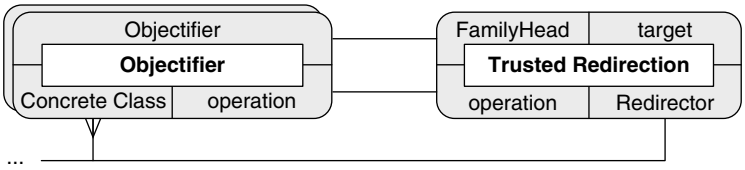

Figure 4.15 *Object Recursion* as just PIN.

We composed the top half of the *Decorator* pattern and have just one piece left to go to finish our definition. The remaining concept is *Extend Method*, which fills out the bottom of *Decorator* by extending *Trusted Redirection*, as shown in Figures 4.16 and 4.17. *Extend Method*'s original behavior and recursor from *Object Recursion* merge into *Decorator*, tying these two smaller patterns together.[4]

We can simplify this by reducing it again to just the PINboxes, as in Figure 4.18. Furthermore, we can always wrap and reduce this to a single PINbox indicating an instance of *Decorator*, as in Figure 4.19. The role names here are taken directly from the participants section of the *Decorator* specification in the Gang of Four (GoF) text [21], with two additions: operation and componentObj. These are implicitly discussed in the participants section, but we make them explicit here to clarify the pieces involved.

Now we have a simple, concise notation for the *Decorator* pattern. At this point, however, we can go the other way as well. We can use expanded PINboxes to increasingly expose finer granularity in *Decorator* by showing the underlying hierarchy of concepts. Figure 4.20 shows *Decorator* as an expanded PINbox, revealing its direct internal wiring. Figures 4.21 and 4.22 drill into *Object Recursion* and *Objectifier*, respectively, and finally Figure 4.23 expands *Fulfill Method* to the EDP level, at which point we can decompose no further. *Decorator* is now fully revealed. Each of these diagrams is equivalent to the others.

So that's *Decorator*, and we built it with just four EDPs: *Abstract Interface*, *Inheritance*, *Trusted Redirection*, and *Extend Method*. Each is a simple concept, but together, linked in a very specific combination, they describe a fairly high-level abstraction that is commonly found in software systems. What's more, we demonstrated that interim concepts can be studied and mastered to gain a more thorough understanding of *Decorator*.

4. `ConcreteDecoratorA` and `Decorator` form another leg of *Inheritance*, but we're leaving it out for clarity at the moment.

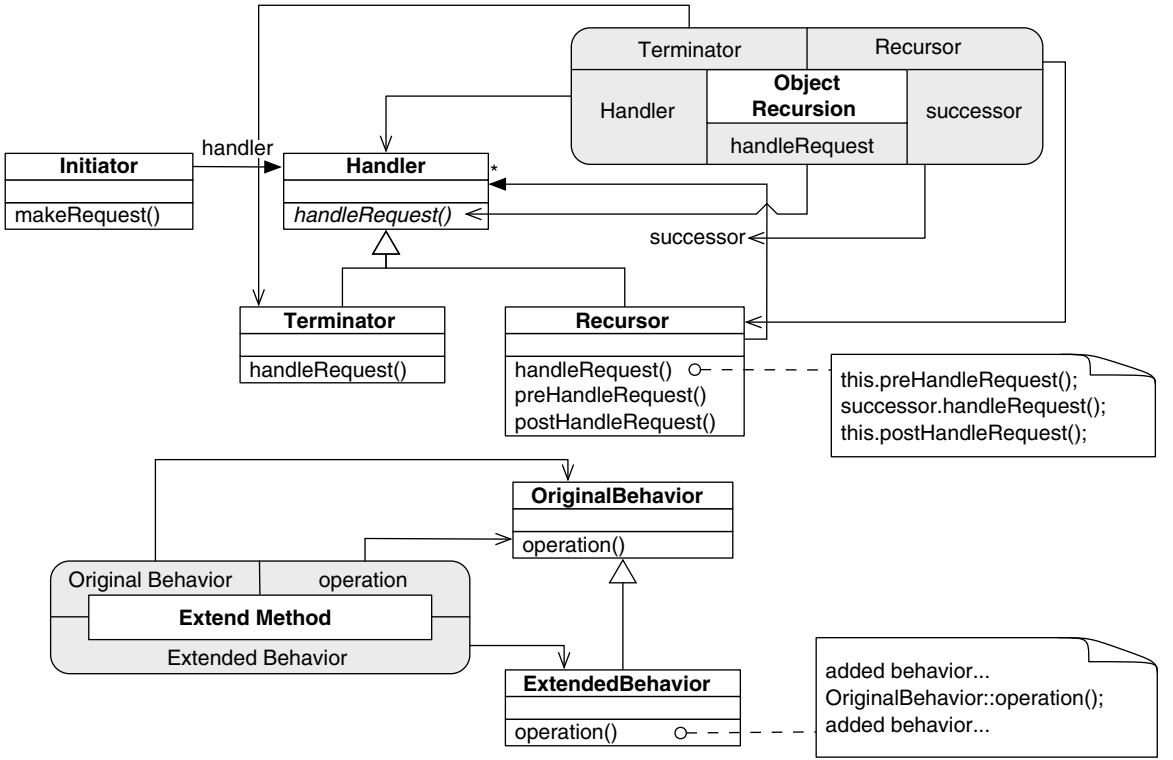

Figure 4.16 *Object Recursion* and *Extend Method*.

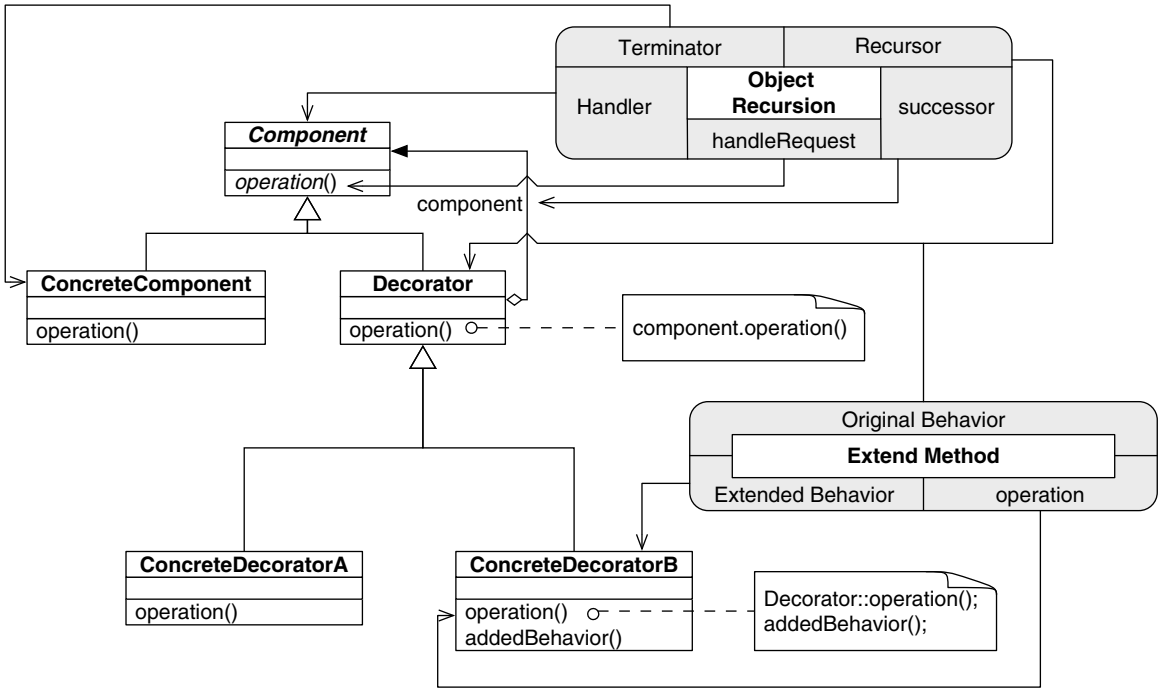

Figure 4.17 *Decorator* annotated with PIN.

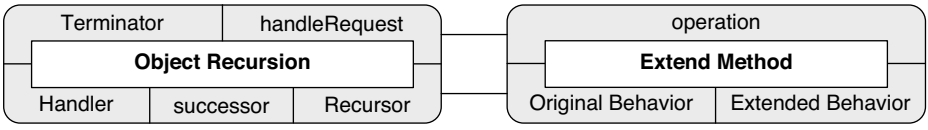

Figure 4.18 *Decorator* as PIN.

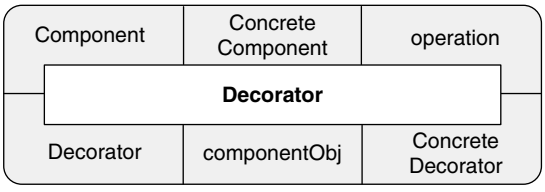

Figure 4.19 *Decorator* instance as a PINbox.

Most important, we never once discussed code. We didn't bring up classes, or methods, or fields. We talked about concepts and ideas only, yet we achieved a framework that provides guidance and precision.

In EDPs, we have the building blocks with which to form great software that we *understand*.

The final section of the GoF text [21, p. 358] uses a quote from Christopher Alexander to describe what is "good design." I can think of no more fitting description of their own design patterns when viewed as dovetailed and intertwined examples of the EDPs.

> It is possible to make buildings by stringing together patterns, in a rather loose way. A building made like this, is an assembly of patterns. Is it not dense. It is not profound. But it is also possible to put patterns together in such a way that many patterns overlap in the same physical space: the building is very dense; it has many meanings captured in a small space; and through this density, it becomes profound. [3, p. xli]

The design patterns literature is full of profundity through the composition of smaller concepts in dense and precise ways. EDPs let us view that density with clarity and insight.

One more comment. Do you recall the example from the beginning of Chapter 2 about the hidden *Decorator* pattern in industrial code that inspired SPQR? SPQR, using the EDPs, composition technique, and formalisms described here, was able to identify its existence in just a couple of seconds without being given any hints. Compared to almost 200 hours, that's a lot of coffee breaks that could be taken instead.

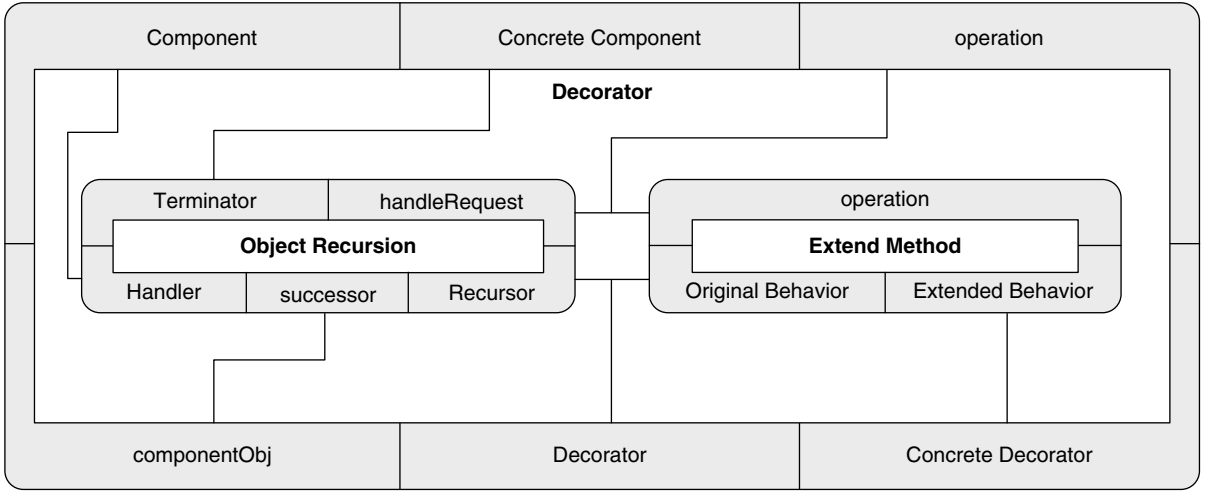

Figure 4.20 Expanding *Decorator*: one level.

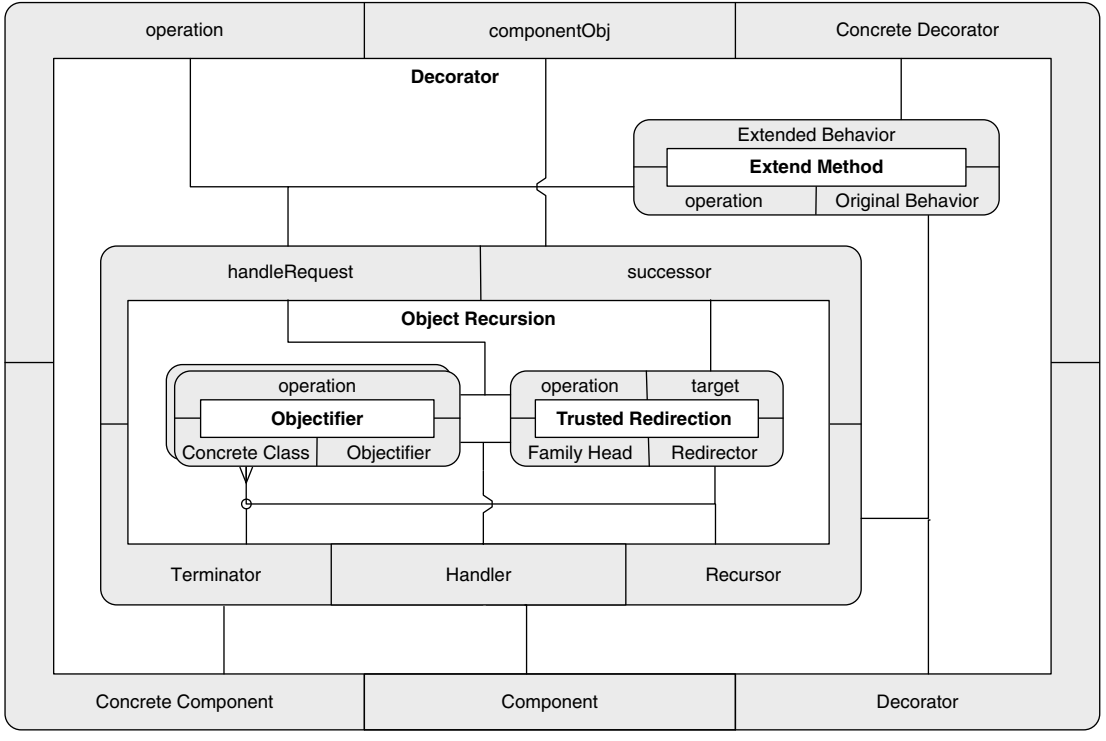

Figure 4.21 Expanding *Decorator*: two levels.