Extendibility: Adding Operations

The argument for object technology's support for extendibility comes in part (in addition to mechanisms such as information hiding and genericity, as well as the central role of contracts) from the assumption that the most significant changes in the life of a system are of the kind just discussed: introducing a type that shares some operations with existing types and may require new operations. Experience indeed suggests that this is the most frequent source of nontrivial change in practical systems, where object-oriented techniques show their advantage over others. But what of the other case: adding operations to existing types? Some client application relying on the notion of pudding might, for example, want to determine the `cost` of making various puddings, even though pudding classes do not have a `cost` feature.

Functional programming performs neither better nor worse for the addition of an operation than for the addition of a type: it's a matter of adding 1 to `f` rather than `t`. The object-oriented solution, however, does not enjoy this neutrality. The basic solution is to add a feature at the right level of the hierarchy. But this has two potential drawbacks:

- Because inheritance is a rather strong binding ("is-a") between classes, all existing descendants are affected. In general, adding a feature to a class at a high position in the inheritance structure can be a delicate matter.
- This solution is not available if the author of the client system is not permitted to modify the original classes, or simply does not have access to their text—a frequent case in practice since these classes may have been grouped into a library, for example, a financial contract library. It would make no sense to let authors of every application using the library modify it.

Basic object-oriented techniques (e.g., Meyer 1997) do not suffice here. The standard OO solution, widely used, is the *visitor pattern* (Gamma et al. 1994). The following sketch, although not quite the standard presentation, should suffice to summarize the idea. (It is summarized from Meyer's *Touch of Class: An Introduction to Programming Well* [2008], a first-semester introductory programming textbook—suggesting how fundamental these concepts have become.) Figure 13-5 lists the actors involved in the pattern.

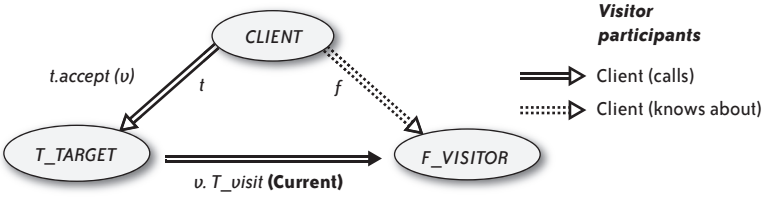

FIGURE 13-5. Actors of the Visitor pattern