Emacs provokes such strong reactions partly because there's so much of Emacs to react to. The current Emacs sources include 1.1 million lines of code written in Emacs's own programming language, Emacs Lisp. This corpus includes code to help you edit programs in C, Python, and other languages, as you might expect from a programmer's text editor. But it also includes code to help you debug running programs, collaborate with other programmers, read electronic mail and news, browse and search directories, and solve symbolic algebra problems.

Looking under the hood, the story gets stranger. Emacs Lisp has no object system, its module system is just a naming convention, all the fundamental text editing operations use implicit global arguments, and even local variables aren't quite local. Almost every software engineering principle that has become generally accepted as useful and valuable, Emacs flouts. The code is 24 years old, huge, and written by hundreds of different people. By rights, the whole thing should blow up.

But it works—and works rather well. Its bazaar of features grows; the user interface acquires addictive new behavior; and the project survives broad changes to its fundamental architecture, infrequent releases, leadership conflicts, and forks. What is the source of this vigor? What are its limitations?

And finally, what can other software learn from Emacs? When we encounter a new architecture that shares some goal with Emacs, what question can we ask to gauge its success? Over the course of this chapter, I'll pose three questions that I think capture the most valuable characteristics of Emacs's architecture.

Emacs in Use

Before discussing its architecture, let's look briefly at what Emacs is. It's a text editor that you can use much like any other. When you invoke Emacs on a file, a window appears, displaying the file's contents. You can make your changes, save the revised contents, and exit. However, Emacs is not very effective when used this way: it is slower to start than other popular text editors, and its strengths don't come to bear. When I need to work in this fashion, I don't use Emacs.

Emacs is meant to be started once, when you begin work, and then left running. You can edit as many files as you like within one Emacs session, saving changes as you go. Emacs can keep files in memory without displaying them, so what you see reflects what you're working on at present, but your other tasks are ready to be called up just as you left them. The experienced Emacs user closes files only if the computer seems to be running low on memory, so a long-running Emacs session may have hundreds of files open. The screenshot in Figure 11-1 shows an Emacs session with two frames open. The left frame is split into three windows, showing the Emacs splash screen, a browsable directory listing, and a Lisp interaction buffer. The right frame has a single window, displaying a buffer of source code.