listings, and the like all go into buffers with appropriately chosen names. This may seem like a cheap implementation trick—it does simplify Emacs internally—but it's actually quite valuable because it means that these different kinds of content are all ordinary editable text: you can use the same commands to navigate, search, organize, trim, and sort this data that are available to you in any other text buffer. Any command's output can serve as any another command's input. This is in contrast with environments such as Microsoft Visual Studio, where the results of, say, a search can only be used in the ways the implementors anticipated would be useful. But Visual Studio is not alone in this; most programs with graphical user interfaces have the same shortcoming.

For example, in the screenshot, the middle window of the frame on the left shows a directory listing. Like most directory browsers, this window provides terse keyboard commands for copying, deleting, renaming, and comparing files, selecting groups of files with globbing patterns or regular expressions, and (of course) visiting the files in Emacs. But unlike most directory browsers, the listing itself is plain text, held in a buffer. All the usual Emacs search facilities (including the excellent *incremental search* commands) apply. I can readily cut and paste the listing into a temporary buffer, delete the metadata on the left to get a plain list of names, use a regular expression search to winnow out the files I'm not interested in, and end up with a list of filenames to pass to some new operation. Once you've gotten used to working this way, using ordinary directory browsers becomes annoying: the information displayed feels out of reach, and the commands feel constrained. In some cases, even composing commands in the shell can feel like working in the dark because it's not as easy to see the intermediate results of your commands as you go.

This suggests the first question of the three I promised at the beginning of this chapter, a question we can ask of any user interface we encounter: *how easy is it to use the results of one command as input to another?* Do the interface's commands compose with each other? Or have the results reached a dead end once they've been displayed? I would argue that one of the reasons many programmers have been so loyal to the Unix shell environment, despite its gratuitous inconsistencies, anorexic data model, and other weaknesses, is that nearly everything can be coaxed into being part of some larger script. It's almost easier to write a program that is readily scriptable than not, so dead ends are rare. Emacs achieves this same goal, although it takes a radically different route.

Emacs's Architecture

Emacs's architecture follows the widely used *Model-View-Controller* pattern for interactive applications, as shown in Figure 11-2. In this pattern, the *Model* is the underlying representation of the data being manipulated; the *View* presents that data to the user; and the *Controller* takes the user's interactions with the View (keystrokes, mouse gestures, menu selections, and so on) and manipulates the Model accordingly. Throughout this chapter, I'll capitalize Model, View, and Controller when I'm referring to elements of this pattern. In Emacs,