class **MenuMorph** and **GraphicalDictionaryMenu**. **MenuMorph**s understand some Morphic-specific features, like **addStayUpItem** (which allows a menu to stay available for later mouse clicks). When a **MenuMorph** is being constructed, it is also possible to specify **balloonTextForLastItem:** to set up help for users. **GraphicalDictionaryMenu** knows how to display forms for items, which can be a useful alternative in many situations.

EXERCISES: WORKING WITH PLUGGABLE INTERFACES

5. Redesign the **ClockWindow** so that there is no Stop button and the **Clock** is stopped as soon as the **ClockWindow** is closed.
6. Get rid of the **ClockWindow** class and make the user interface work from **Clock**.
7. Use pluggable components to make a simple Rolodex. Create Rolodex cards that contain name, address, and phone-number information. Provide a scrolling list of names, and when one is selected, display the information in a text area.
8. Use pluggable components to make a simple calendar system. Provide a multi-pane list browser for picking a year (within, say, a ten-year range), a month, and a date. (Be sure to fill in the date pane only when the year and month are selected!) Allow the user to fill in text-pane notes for the given date. Use a **Dictionary** to store the text information, with the dates as the indices.

START HERE!

5.4 BUILDING MORPHIC USER INTERFACES

The real strength of Morphic lies in the creation of Morphic interfaces within Morphic. Morphic interfaces don't necessarily have to follow the MVC paradigm, but they can. Morphic interfaces can also be assembled rapidly through simple dragging and dropping. We have already seen that one morph can be *added* to another. From within Morphic, we say that one morph can be *embedded* within another.

In this section, we'll explore how to work with morphs from the user-interface perspective, and then from the programmer's perspective. We'll use the same example, a simple simulation of an object falling, to explore both sides. Along the way, we'll describe the workings of Morphic.

5.4.1 Programming Morphs from the Viewer Framework

The Viewer framework (sometimes called the *etoys system*) has been developed mainly by Scott Wallace of the Disney Imagineering Squeak team as an easy-to-use programming environment for end users. It's not a finished item, and it may change dramatically in future versions of Squeak. But as-is, it provides us a way of exploring Morphic before we dig into code.

We're going to create a simulation of an object falling. Our falling object will be a simple EllipseMorph. Our falling object will have a velocity (initially zero) and a constant rate of acceleration due to gravity. We'll use pixels on the screen as our distance units.

If you recall your physics, the velocity increases at the rate of the acceleration constant. For our simulation, we'll only compute velocity and position *discretely* (i.e., at fixed intervals, rather than all the time, the way that the real world works). Each time element, we'll move the object the amount of the velocity, and we'll increment the velocity by the amount of the acceleration. This isn't a very accurate simulation of a falling object, but it's enough for demonstration purposes.

For example, let's say that we run our discrete simulation every second. Let's say that the velocity was currently 10 and the acceleration was 3. We say that the object is falling 10 pixels per second, with an acceleration of 3 pixels per second per second (that is, the velocity increases by 3 pixels per second at each iteration, which occurs every second). When the next second goes by, we add to the velocity so that it's 13 pixels per second, and we move the object 13 pixels (because that's the velocity). And so on.

We'll also create a *Kick* object. When the object is kicked, we'll imagine that the object has been kicked up a few pixels, and its velocity goes back to zero. Strictly speaking, an upward push on the falling object would result in an upward velocity that would decrease as gravity pulled the object back down. (This is simplification for the sake of a demonstration.)

Create three morphs (from the *New Morph* menu, or from the Standard Parts bin, or from the Supplies flap): A **RectangleMorph** (default gray), an **EllipseMorph** (default yellow), and a **TextMorph** (appears in Supplies and Parts as "Text for Editing"). We're going to use the rectangle and text as our Kicker, and the ellipse as our falling object.

We'll start out by creating our Kicker button. Click on the text so that you can edit it, and change it to say "Kick." Now Morphic-select it, and drag it (via the black *Pick Up* halo) into the rectangle (Figure 5–19). Use the control-click menu to *embed* the text into the rectangle. After you choose the *embed* menu item, you will be asked to choose which morph you want to embed the text into. Choose the **RectangleMorph**. (As we'll see later in this chapter, the other option, a **PasteUpMorph**, is actually the whole Morphic world. It is possible to embed morphs into the desktop of a Morphic World.) Once embedded, they move as one morph (Figure 5–20).

Now, let's start programming our two morphs. Morphic-select the ellipse and choose the center-left (turquoise) halo, the *View me* halo. When you do, a *Viewer* for the ellipse will open (Figure 5–21).

The Viewer is a kind of browser on a morph. It allows you to create methods for this morph, create instance variables for the given morph, and directly manipulate the morph. Click on one of the yellow exclamation points—the command, whatever it is (say, *Ellipse forward by 5*), will be executed, and the morph will

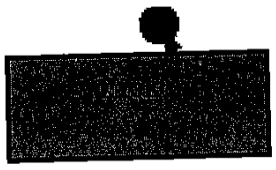

Figure 5–19 Dragging the TextMorph into the RectangleMorph