Application facade

There's a classic pitfall in building a strong domain model. The presentation layer—or in this case, the UI Model—often gets too intimate with the domain model. If the presentation traverses relationships in the domain, then it becomes difficult to change the domain model. Like any agile team, we needed to stay flexible, and there was no way we would make design choices that would lead to less flexibility over time.

Martin Fowler's "Application Facade" pattern fit the bill (see the "References" section at the end of this chapter). An application facade presents only a portion of the domain model to the presentation layer. Instead of walking through graphs of domain objects, the presentation asks the application facade to assist with traversal, life cycle, activation, and so on.

Each form defined a corresponding facade interface. In fact, following the dictum that consumers—rather than their providers—should define interfaces we put the facade interface in the form's package. The form asks the facade to look up domain objects, relate them, and persist them. In fact, the facades managed all database transactions, so the forms were never aware of transaction boundaries.

The interfaces at this boundary, between forms and facades, also became an ideal place to isolate objects for unit testing. To test a particular form, the unit test creates a mock object that implements the facade's interface. The test trains the mock object to feed the form with some set of expected results, including error conditions that would be very difficult to reproduce with the real facade. I think we all regarded mock objects as a two-sided compromise: although they made unit tests possible, something still felt wrong about tying the tests so closely to the forms' implementations. For example, mock objects have to be trained with the exact sequence of method calls to expect, and the exact parameters. (Newer mock object frameworks are more flexible.) As a result, changes in the internal structure of the forms would cause tests to fail, even though no externally visible behavior changed. To a certain extent, this is just the price you pay for using mock objects.

All the Creation Center applications, both in the studio and in the printing facility, used the same stack of layers. Removing the GUI from the driver's seat kept the team from spending endless cycles in Swing tweaking. This inversion of control also provided a uniform structure that every application, and every pair, could follow. Even though we created more than the usual "three-layer cake," our stack was quite effective at separating concerns: Swing was limited to the UI, domain interaction in the forms, and persistence in the facades.

Interchangeable Workstations

When a photographer finishes a session, she grabs any open workstation. Depending on how busy the studio is, she'll usually finish with the customer at that time. It's common, though, for customers to come back later, maybe even on a different day. It would be ridiculous to permanently attach a customer to a single workstation—not just unworkable for scheduling, but also risky. Workstations break!