Editors: Andy Hunt and Dave Thomas ■ The Pragmatic Programmers
andy@pragmaticprogrammer.com ■ dave@pragmaticprogrammer.com

Naked Objects

Richard Pawson

On a recent visit to Norway, I was served the customary lunch of open-faced sandwiches. I don't understand the logic of the open-faced sandwich—all the work of preparing a sandwich, but you still have to eat it with a knife and fork. Perhaps the Norwegian that brought the idea of the sandwich from overseas (our British tradition holds that the Earl of Sandwich invented it) simply lost the key instruction: place another slice of bread on top so that you can pick it up!

Some Norwegians might feel the same way about what the rest of the world did to one of their great ideas: object-orientation. Looking at most so-called object-oriented software, you'd think the designers had never heard about the key idea: An object should model some aspect of the business domain—not just its attributes and associations—but all behaviors that might be required of it. This behavioral completeness yields a big benefit, in that changes to the business domain map simply to changes in the software. Actually, many designers intend to create behaviorally complete objects, but just as milk degrades into curds and whey and salad dressing into oil and vinegar, their software seems almost inevitably to separate into process and data.

Separating process and data

Several years ago I determined to find out what caused this. My research identified several common practices that encourage the separation of process and data in object-oriented designs. The problem is that none of those practices is a bad habit; each was deliberately designed to overcome some existing problem with software engineering.

Let's look at an example. The argument for the Model-View-Controller pattern and its close-relation, the Entity-Boundary-Controller pattern is that a given business object must be viewed on different platforms, in different contexts, and with different visual representations. Embedding knowledge of how to create these different views in business objects would result in bloated objects with heavy duplication of functionality. Using MVC or EBC, the core business objects (that is, the Model or Entity objects, respectively) have no knowledge of these different presentations. Dedicated presentation objects (that is, the View or Boundary objects) specify what will appear in each view, and know how to create the visual display and accept inputs from the user. `Controller` objects provide the glue between the two—populating the views with attributes from the business objects and invoking methods on those objects in response to user-initiated events.

However, there is a marked tendency for `Controller` objects to become an explicit representation of business tasks, managing the optimized sequence of activities and enforcing appropriate business rules. In doing so, they usurp the business objects' natural responsibilities. The presentation objects, meanwhile, often mix attributes from multiple business objects. Before long, the `Entity` or `Model` objects become the data representation, and the Controller objects become the process representation.

0740-7459/02/$17.00 © 2002 IEEE

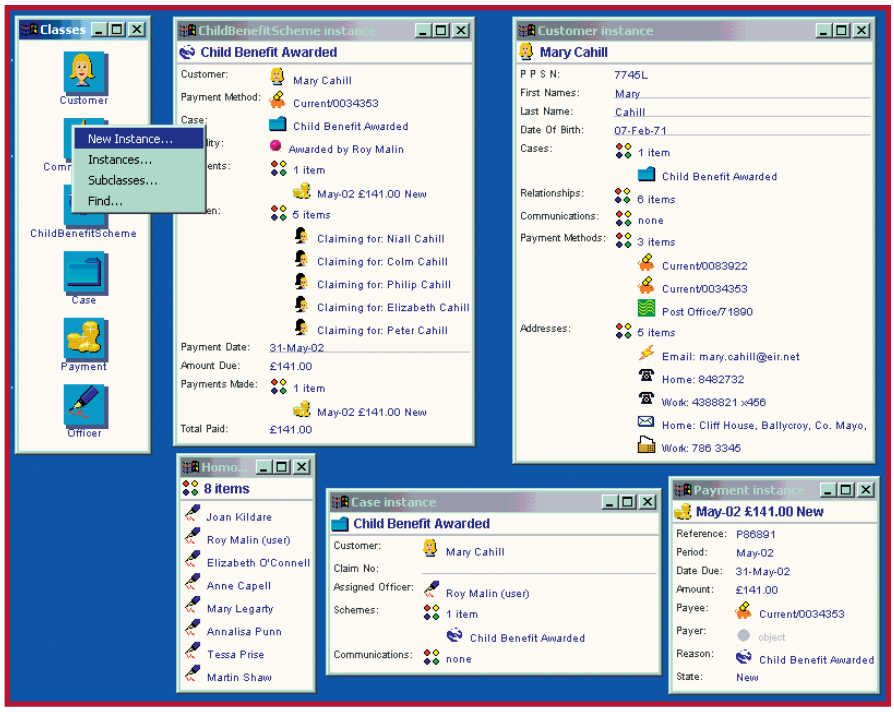

Figure 1. A prototype for a new government benefits administration system built using Naked Objects. All user actions consist of noun-verb style operations on core business objects.

Naked Objects

Robert Matthews and I designed Naked Objects, an open-source Java-based framework, to discourage process and data separation and to encourage business system creation from behaviorally complete objects. Naked Objects' key idea is to directly expose the core business objects (such as `Customer`, `Product`, and `Order`) to the user. All user actions consist of invoking a behavior directly on a business object, either by selecting an icon from a pop-up menu or dragging an object and dropping it on another. There are no top-level menus, or any forms or scripts sitting on the objects. The framework is so radical that it does not even permit the creation of dialog boxes.

Applying the principles of Naked Objects

One of the first organizations to apply the design principles embodied in Naked Objects is the Irish government's Department of Social, Community, and Family Affairs. Figure 1 shows a screenshot from a prototype for a new system, created using Naked Objects, to replace their existing Child Benefit Administration system. That system is now in full-scale development and is near deployment. Although the user interface differs slightly from that shown here, the prototype's principles have been faithfully preserved. The new architecture, which mimics that of the Naked Objects framework but is tailored to the DSCFA's own technical environment, will eventually replace all of their existing core business systems.

Figure 1 shows several classes of business objects including `Customers` (the mother and children involved in a claim), `Officers`, `Cases`, `Payments`, `Communications`, and `Schemes` (the various legislative schemes that the DSCFA administers—only `Child Benefit` is included in this prototype), which are represented as icons. Users can double click on any instance to show an expanded view, or they can drag it to the desktop to create a separate viewer. Right clicking on any instance, in any context, produces a pop-up menu of actions, some of which are generic and some of which are object specific. These actions all correspond to methods on the business object. For a `Customer` object, specific actions include `Authenticate`, `Communicate`, and `RegisterNewChild`. Some actions perform simple operations on that object; others initiate more complex transactions.

Users can also initiate behaviors by dragging and dropping objects on other objects, or into specific fields within objects. Dragging a `Customer` instance on the `Child Benefit` class would cause the system to check if that customer was already claiming child benefit. If not, the system initiates a new claim by creating an instance of the `Child Benefit Scheme`, inserting a reference to the `Customer` object inside it, and creating a new instance of `Case` to track the progress of this claim. The `Case` owner is the object representing the `Officer` logged into the system. (The user could also perform each action individually and in a different order).

The system also portrays the business object classes directly. A window on the left lists the six primary business object classes that constitute the system. Each class icon has a pop-up menu of class methods. These are mostly generic methods, and include

- Create a new instance of this class
- Retrieve an existing instance from storage
- List instances of this class that match a set of criteria
- Show any subclasses that are available to the user

Because there is no other mechanism with which users can initiate a business action, the programmers developing an application are forced to encapsulate all required behavior with the core business objects. However, because there is a 1:1 correspondence between the underlying object model and the user interface, Naked Objects can auto-generate the latter from the former. Programmers do not need to write any user interface code to develop a fully working

application using Naked Objects. Instead, at runtime the framework's generic viewing mechanism uses Java's reflection to identify objects' behaviors and makes these available to the user. Programmers need only follow a few simple conventions when writing business objects in Java. The Naked Objects framework effectively provides the `View` and `Controller` roles in generic form—the programmer writes only the `Model` objects.

Most early attempts at auto-generating a user interface from an underlying model, such as some application generators in the 1980s, produced spectacularly poor user interfaces. Naked Objects does rather better. I would not suggest that our user interfaces are in any sense optimal, but they are remarkably expressive and give users a strong sense of control—what Edwin Hutchins, James Hollan, and Donald Norman call "direct engagement"[1] and Brenda Laurel calls a "first-person experience."[2] We would even suggest that much of the efforts of the human-computer interface community, which is concerned with optimizing a user interface design for a given set of tasks, is actually counterproductive. Who has not experienced the frustration of being managed by an optimized script when dealing with a customer agent at a call center, for example?

I don't mean to suggest that the user is free to do anything with a Naked Object system—some constraints are genuinely needed. But these constraints are built into the business objects, not into the scripted tasks that usually sit on top of them. The net effect is similar to using a drawing program or a CAD system, but for business transactions.

When developing an application using Naked Objects, the 1:1 correspondence between the screen representation and the underlying model means that the business objects now provide a common language for the user and programmer. The auto-generation of one from the other shortens the development cycle to the point where we can now create prototypes in real time. Most approaches to rapid prototyping develop only the users' view of the system. Using Naked Objects we are prototyping the object model simultaneously. With Naked Objects, the conventional notion of the user interface simply disappears—programmers don't even consider a user-interface because they don't have to develop one and users talk only about the objects with which they deal.

New generation of business systems

A handful of commercial organizations have already started to use Naked Objects to conceive a new generation of business systems. In each of these projects, the organization has stated that they have never seen such positive communication between developers and users during development. Developers have praised Naked Objects for its "flexibility"—quite an interesting comment given that the framework doesn't let them do many things they are accustomed to doing, such as customizing the user interface or writing dialog boxes. (The latter is a massive constraint and somewhat difficult to adjust to, but we have found that it forces you to do a better object modeling job.) Programmers frequently comment, "This feels like what I had in my mind when I first encountered OO, and I've just realized how far it is from what we've actually been doing."

Naked Objects is still in its infancy, but it is developing rapidly. We have much that we still want to do with the framework. We are starting to get offers of help from others who are catching the vision. What pleases us most is that the framework gets simpler with every iteration. You can read more about the design principles behind Naked Objects and download the Java framework from www.nakedobjects.org.

Antoine de Saint Exupery wrote that "In anything at all, perfection is finally achieved not when there is no longer anything to add, but when there is no longer anything to take away, when a body has been stripped down to its nakedness." This principle has been one of the inspirations for Naked Objects. I remain to be convinced about the merit of naked sandwiches, though.

"In anything at all, perfection is finally achieved not when there is no longer anything to add, but when there is no longer anything to take away, when a body has been stripped down to its nakedness."

References

1. E. Hutchins, J. Hollan, and D. Norman, "Direct Manipulation Interfaces," *User Centered System Design: New Perspectives on Human-Computer Interaction*, D. Norman and S. Draper, eds., Lawrence Erlbaum, Hillsdale, N.J., 1986, p. 94.
2. B. Laurel, *Computers as Theatre*, Addison-Wesley, Reading, Mass., 1991, pp. 116–119.

Richard Pawson is a research fellow with Computer Sciences Corporation. He has a BSc in engineering science and is currently earning a PhD in computer science at Trinity College, Dublin. He is coauthor of *Naked Objects* with Robert Matthews, which will be published by Wiley in November 2002. Contact him at CSC Research Services, 70 Wilson St., London EC2A 2DB, UK; rpawson@csc.com.