

# **Naked Objects – Do You Really Need User Interfaces?**

James W. Cooper

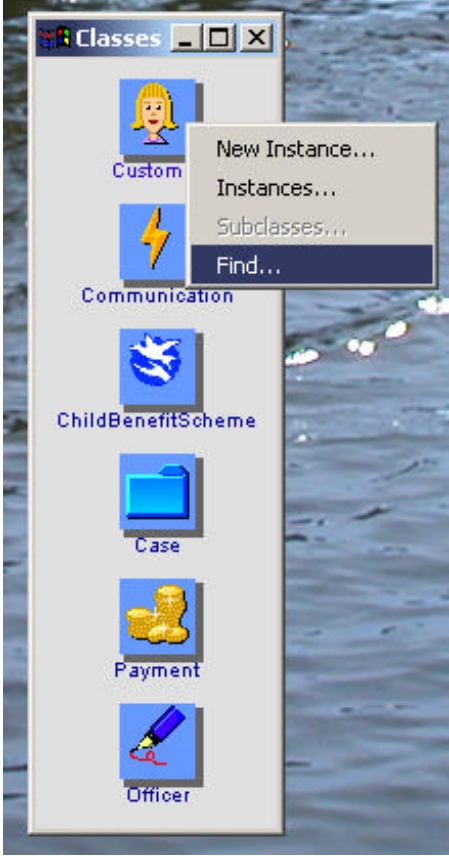
One of the most thought-provoking talks and demonstrations I heard while attending the OOPSLA Conference in October of 2001, was a paper by Richard Pawson, based on his work with Robert Matthews on building user applications without conventional user interfaces.

In this work, they undertook a study to provide a way for end users of an application to carry out operations directly on the objects that make up the application. This radical new approach almost completely does away with the concept of an application with a user interface. Instead, you work directly with the objects, each of which has a similar minimal inspection interface, and you cause interactions between objects by simple dragging and dropping actions.

In order for this approach to work, they spend a good deal of time defining the exact nature of the objects in an application by working directly with the customers or users to find out what their view of the activity is. Generally, they can begin to define the objects they need to use after only a couple of days of meetings. This means that prototyping and design activities devolve to defining the objects and their methods in simple terms that correspond to the customer's view of the system. For example the objects in a company billing system might be Customers, Products, Orders and so forth.

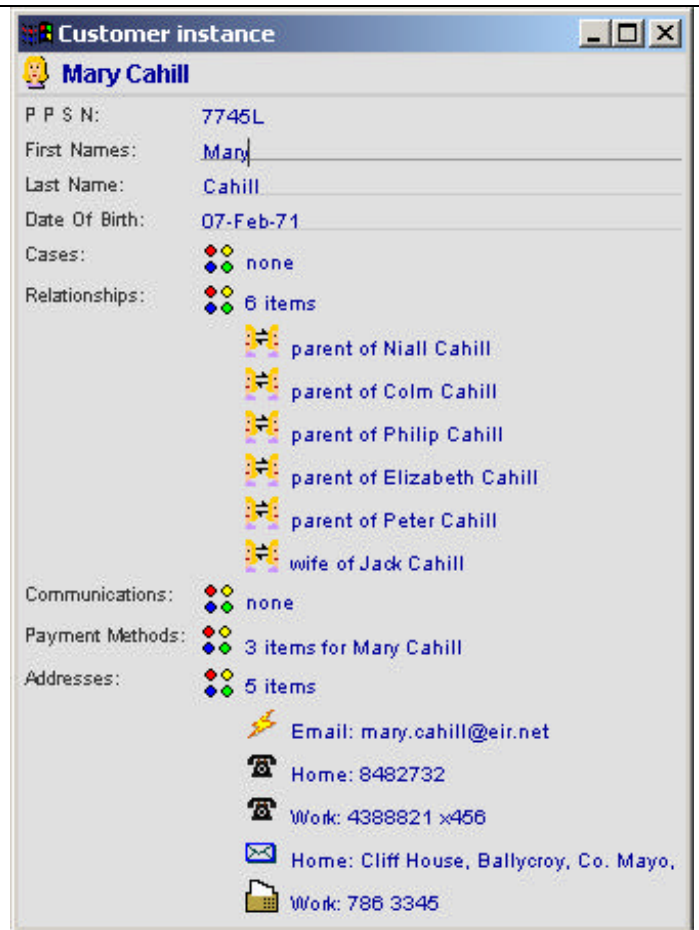
This approach is radical not only because of the minimalist user interface, but because of the way such a system empowers the user. Since the user can see exactly how the system works, working with the system is extremely straightforward. In addition, since the user interface is basically a visual representation of the objects themselves, the system provides a common language between the users and the developers. This makes requirements gathering and prototyping a great deal simpler to carry out.

## A Naked Experience

	<p>Pawson, Matthews and Simon Dobson reported on their experience in creating a system for handling child benefits for the Irish Department of Social, Community and Family Affairs. The screenshots I show here are from a downloadable demo. The actual system went into production early this year.</p> <p>When you start their benefits program, you are presented with a panel containing 6 objects (Figure 1).</p> <p>As you can see in Figure 1, the 6 objects as Customer, Communication, ChildBenefitScheme, Case, Payment and Officer.</p> <p>You log in as a case officer by selecting your name from the instances of the Officer object.</p> <p>You can examine instances of each object by right clicking on it or by dragging it onto your desktop. If there are a large number of instances of an object, you can use the pop-up menu's Find action to locate a particular instance, or by bringing up a list of instances as shown in Figure 2.</p> <p>Once you find a specific customer you can drag that name onto your desktop and see the details as shown in Figure 3.</p>
<p><b>Figure 1 – The basic objects</b></p>	

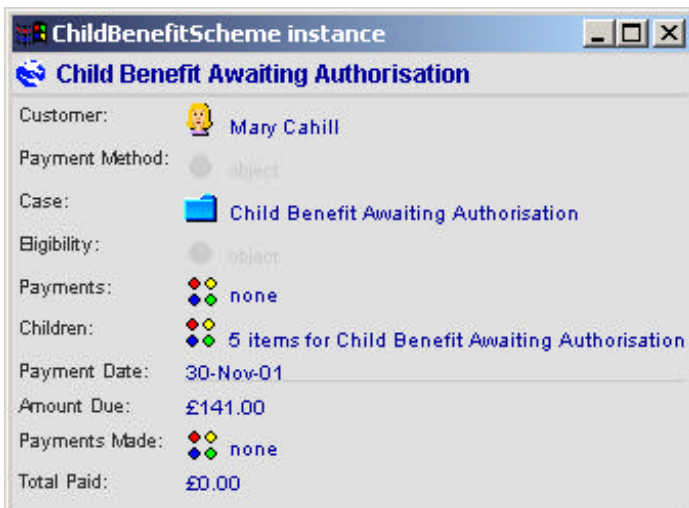


**Figure 2- A list of the instances of the Customer object.**



**Figure 3 – A specific customer (parent) with her relationships shown.**

Using a similar drag and drop mechanism, you can create a new instance of a ChildBenefitScheme by dragging one onto your desktop, and assign it to the customer Mary Cahill by dragging her name over the benefit scheme object, as shown in Figure 4.



## **Figure 4 – A new benefit scheme for Mary Cahill**

So, as you can see, you can view the objects of this application, view the objects they contain, view their methods, and act on them with a simple drag and drop mechanism. While the system is not without its own user interface, you can appreciate that each object has a similar interface and that you can learn to use it quickly. There are no tedious panels of tabbed information to wander through to get to the meat of the application.

### ***How Did They Write This?***

This entire application is written in Java, and makes use of the Java reflection API to find out the methods of each object. Since reflection takes place at run time, each object is entirely independent of the others and the actual methods can change over time without introducing any inter-object dependencies.

Best of all, the entire toolkit, written primarily by Robert Matthews, is available free for download under the Gnu public license from [nakedobjects.org](http://nakedobjects.org). So, not only can you download and play with this demo, you can write real code yourself using this object framework. The toolkit includes their Object Viewing Mechanism (OVM) that produces the neat displays you see in Figures 1-4.

### ***What Did Customers Think?***

In their paper (1, 2), the authors report that senior management immediately recognized that everyone in the entire organization “right up to the Government Minister” could use the same system. While there was some worry that this interface might not be the best one for repetitive data entry, there is little such work remaining in this agency after automation of the past few years.

Probably the most important result is that this sort of interface is reported to reinforce the message that the government works are “problems solvers rather than process followers.” And this is a very significant improvement for any such agency.

The authors reported that user picked up the style of this sort of program system in just a few minutes and became conversant with the object lingo very shortly thereafter, and that they liked the system because it “gave them a lot of freedom to work in the way that suited them best.”

There are several possible objections to such a system, that the authors also deal with. One obvious one is that it is not clear that government workers should have this much freedom to make decisions and that some business rules might be inadvertently violated. In fact, the business rules become part of the individual objects in this system, and there is no more freedom than any other style of system would provide, but it appears to me that there is a great deal more flexibility.

Another possible concern is that of “bloated objects,” where each object takes on too many methods and becomes difficult to use and understand. This is a serious concern in *any* sort of OO system, and the solution is the same regardless of whether the objects are naked or clothed. In either case, you need to refactor the system to move methods into more appropriate objects.

The authors note that among the users of the toolkit, many have cited the “flexibility” it gives them. The note this with some surprise, since Naked Objects places so many restrictions on developers. You can’t have any scripts or dialogs or screens or form layouts. In fact, Naked Objects represents the complete disappearance of the user interface! In spite of these restrictions, this is a very powerful system that should gain more attention as a way to write powerful, elegant usable programs. I certainly plan to experiment with it.

## Writing Your Own

I got some example code from Robert Matthews showing how you write NakedObjects code systems, since the website was at this writing still a bit incipient. A demonstration of a NakedObjects flight booking system is shown in Figure 5.



Figure 5 – A Naked Objects flight booking system.

The window that displays these objects is controlled by a simple XML configuration file:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE application SYSTEM "app.dtd">

<application name="ECSI Telephonist">
  <class name="User"/>
  <class name="Customer"/>
  <class name="City" plural-noun="Cities" instantiable="yes"/>
```

```
<class name="Location"/>
<class name="Booking"/>
<class name="Flight"/>
</application>
```

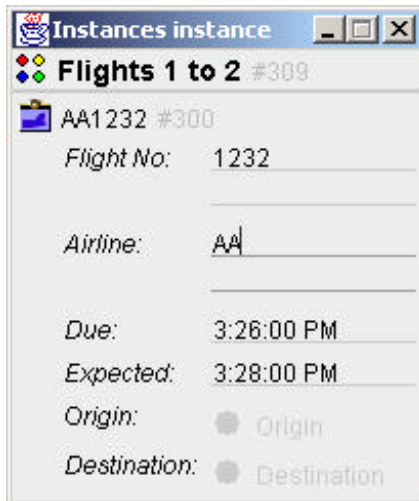
Then, every object you write is derived from NakedObject, as this Flight object is:

```
public class Flight extends NakedObject {
    private final TextString airline;
    private final TextString flightNo;
    private final Date expected;
    private final Date due;
    private Location origin;
    private Location destination;
public Flight() {
```

and every object must have a default constructor. The get and set methods for each object are its attributes, and are displayed when instances of the object are displayed. For example for the Flight object we have

```
// accessors
    public TextString getAirline() {
        return airline;
    }
    public Location getDestination() {
        return destination;
    }
    public Date getDue() {
        return due;
    }
    public Date getExpected() {
        return expected;
    }
    public TextString getFlightNo() {
        return flightNo;
    }
    public Location getOrigin() {
        return origin;
    }
    public void setDestination(Location dest) {
        this.destination = dest;
    }
    public void setOrigin(Location origin) {
        this.origin = origin;
    }
}
```

These are displayed in an object instance display like that in Figure 6.



The actions that occur when you right click on objects and drag objects onto each other are controlled by methods with “action” as part of their name. If the action method has no parameters, it appears in the menu. To create a booking to the flight destination, you would create the following method

```
public Booking actionNewBookingFromFlightDestination() {  
    Booking booking = (Booking)createInstance(Booking.class);  
    booking.setFlight(this);  
    Location dest = this.getDestination();  
    booking.setPickUp(dest);  
    booking.setCity(dest.getCity());  
    booking.objectChanged();  
    return booking;  
}
```

These methods are displayed in the right-click menus of each object instance as shown in Figure 7.

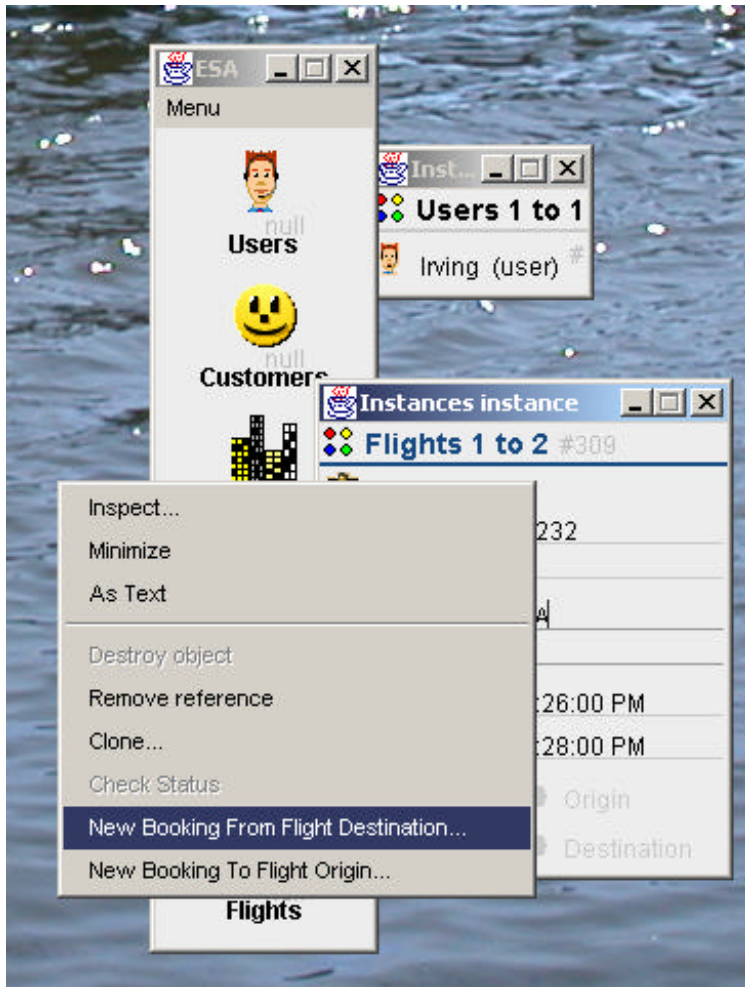


Figure 7 – How to create a new booking for a flight.

If the action method has parameters it is called as a result of dragging on object over another. For example the Booking object has the following action method:

```
public void actionFlightBooking(Flight flight) {
```

which allows you to drag a Flight object onto a booking object.

## Summary

As you can see, this is a very powerful system and it takes only a small amount of code to get it running. I want to thank Robert Matthews for the loan of some example code, and Richard Pawson for a preprint of his paper.

You can download the complete toolkit from <http://nakedobjects.org>. In addition, you can read Pawson's paper in the December 2001 SIGPLAN and his forthcoming book from Wiley. Before they changed their name, they called this project "Expressive systems," and I downloaded the demo from that address (5). They also note that this bears some relationship to work done in the Smalltalk-derived *Squeak* language (6).

## **References**

1. Richard Pawson and Simon Dobson, “Expressive Systems: A radical approach to business system design,” *OOOPSLA 2001 Companion*.
2. Naked Objects, *SIGPLAN, December 2001*.
3. Richard Pawson, *Naked Objects*, John Wiley and Sons, New York: 2002.
4. Toolkit <http://nakedobjects.org>
5. Demo <http://www.expressive-systems.org>
6. D. Ingalls, *et. al.*, “Back to the Future: The Story of Squeak,” in *OOPSLA, 1997*.