Example of OOP Design: Shopping Cart

by David G. Messerschmitt

Supplementary section for Understanding Networked Applications: A First Course, Morgan Kaufmann, 1999.

The process of establishing an application architecture can be illustrated by the on-line book merchant application described in "On-Line Book Selling" in Chapter 3. It was established earlier that a three-tier client-server architecture is natural for this application. Further, this architecture incorporates a Web browser in the client, because most consumers already have this software available and are familiar with using it, and because this contributes to reuse by avoiding a custom development of the client module. The complementary Web server in the application logic tier performs many functions in managing the presentation tier, and also provides a convenient interface to the book merchant-specific functionality in the customer logic through the so-called *common gateway interchange (CGI)*.

The second-tier databases will also use off-the-shelf DBMS components. These components are highly configurable to server this application, including the definition of the number and internal structure of the customer, merchandise, and order databases.

What remains after the choice of these components is the internal decomposition of the customer logic and fulfillment logic on the application logic tier. This will now be illustrated by focusing on one major module, the shopping cart, which manages the process of ordering merchandise. Like the remainder of the on-line merchant application, as described in Chapter 9 the shopping cart underwent the conceptualization and analysis phases, where its basic features and requirements were determined, in consonance with the business objectives like providing the customer with an agreeable shopping experience. Some representative features that were identified include:

- Add a book to purchase.
- Display the current books in the shopping cart, including prices, quantity, and total price. Examine any one of those books in more detail, including reviewing the author, description, cover design, and table of contents.
- Change the quantity of a specific book, including removing it from the shopping cart altogether.
- Initiate an order for all the books currently in the shopping cart, and pay for that order.

Of course, one major requirement is that a large number of customers must be able to create and manage shopping carts, one per customer.

A simplified decomposition of the shopping cart is shown in Figure 1.. There are proxy objects that serve to interface the customer (through the Web server) and to each of the three major databases: merchandise, customer, and order. The remaining objects are integral to the shopping cart itself.

- An object with class `Entry_list` actually manages a list of entries in the shopping cart, and

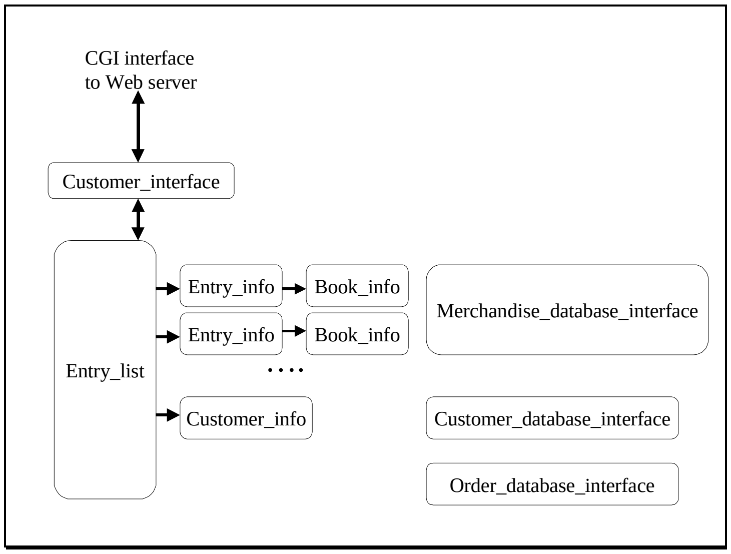

Figure 1. A decomposition of the shopping cart into interacting objects. Arrows indicate some of the more important interactions among objects.

provides basic functions such as adding, modifying, and deleting entries.

- Each entry in the shopping cart is associated with an instance of an `Entry_info` object. This is where the information about the entry is actually stored, such as the quantity and total price. The `Entry_list` object does not keep such information, as an illustration of separation of concerns.
- The merchandise information for each entry is kept in a `Book_info` object. This object keeps and manages information about the book obtained from the merchandise database, such as title and author and unit price. Keeping that information in an object improves performance by avoiding having to return to the database every time the customer views the shopping cart.
- Each `Entry_list` object also manages a `Customer_info` object that keeps tabs on relevant information about the customer who owns that shopping cart, obtained from the customer database.

The interaction among these objects is largely determined by the actions of the customer, as reflected in the interactions initiated by the `Customer_interface` proxy on the customer's behalf. When the customer places the first book in the shopping cart, information about the customer is retrieved from the customer database, a `Customer_info` object is instantiated, and the information stored in it. Similarly, `Book_info` and `Entry_info` objects are instantiated and loaded with the relevant information—the former with author, title, and ISBN for the book, the latter with an indication of the book (including and indication of the `Book_info` object) and the quantity. The `Entry_list` object, which keeps track of all the `Entry_info` objects and allows them to be added and deleted, is instantiated and initialized with an indication of the first `Entry_info` object.

As books are added or deleted or the quantities are changed by the customer, the `Entry_list` object manages those changes at the direction of the `Customer_interface`. Similarly, the Customer_interface can display the current contents of the shopping cart to the customer by retrieving that information from the `Entry_list`, which has to consult the `Entry_info` and

`Book_info` objects in turn.

The object attributes and interface can be illustrated using the `Entry_info` class. Its attributes include everything having to do with one item to be purchased, including the identity and quantity of the book being ordered. Associated with these attributes will be methods allowing them to be set or modified and examined. In addition, it would be convenient to have a method that returns the total price of that item (book price times quantity). It might therefore have the following methods:

```
set_book: Book_info book →;
examine_book: →Book_info book;
set_quantity: Integer number →;
examine_quantity: →Integer number;
examine_price: →Dollars total_price;
```

There are several things to note. Each parameter or return has an associated data type (if it is data) or class (if it is an object). Methods setting an attribute have a parameter but no return, while methods examining an attribute have a return but no parameter. The `examine_price` method does more than set or examine an attribute, it has to calculate the total price as the product of the unit price and the quantity. It will not be able to complete its action without examining the `Book_info` object to determine the unit price.

Suppose another object wanted to know the title of the book being ordered? It would invoke the examine_book method, which would return a `Book_info` object. That `Book_info` object would presumably have a method that returns the title of the book.

While this example conveys the flavor of an OOP architecture, it avoids numerous details in actually programming the application, some of which are touched on in Chapter 11.

Discussion

D1 Discuss some other ways the shopping cart application could be decomposed into interacting objects. Use this to illustrate that there is no single right approach.