CHAPTER 7: MOVING ON TO DESIGN

In the previous installments of the CD Selections case, we saw how Alec, Margaret, and the team had identified the functional and nonfunctional requirements (see Figure 3-A) and had completed the functional (see Figure 4-B, 4C, 4D, 4E, and 4G), structural (see Figures 5-A, 5-B, and 5-D), and behavioral models (Figure 6-B, 6-C, 6-E, and 6-F) of their evolving Web-based solution. However, before they can move into design, they realize that they needed to logically partition the model of the problem-domain. To do this, they have decided to create a package diagram that will represent an overview of the analysis models for their evolving system. In this section of the case, we see how Alec and his team prepared to move from an analysis, or problem-domain, orientation to a design, or solution-domain, orientation. Below, we will see that to get ready for this transition, Alec and his team first create a package diagram to partition the problem-domain layer. Next, they go through a verification and validation of all of the analysis models, and finally, they choose a design strategy to develop the actual design. As in the previous installments of the case, we only deal with the Place Order use case. However, you should remember that object-oriented systems development is holistic. Therefore, to be complete, Alec and his team had to complete the analysis models for all of the use cases associated with the case.

Packages and Package Diagrams

At this point in the development of the Internet Sales System for CD Selections, Alec wants to explicitly partition the evolving system. To do this, Alec decided to use packages to represent both the layers and partitions in each layer. Once he made this decision, he chose to follow the guidelines in Figure 7-21 and the steps for identifying packages and building package diagrams in the textbook. Since, at this point in the development, the team has only been focusing on analysis models, Alec decided that the team should only concentrate on identifying potential partitions on the Problem Domain Layer.

The second step, cluster classes together, was accomplished by reviewing the relationships among the different classes (see Figures 5-D, 6-C, and 6-F). Through this review process, the team saw that there were generalization, aggregation, various associations, and message sending relationships. They also saw the entries in the CRUDE matrix. Since they understood that classes in a generalization hierarchy should be kept together, they clustered the Customer, Person, and Organization classes together to form a partition. Brian pointed out that it is also preferred to keep classes together that participate in aggregation relationships. Based on aggregation relationships, they clustered the Mkt Info, Review, Artist Info, and Sample Clip classes together in a partition. Based on the association relationship and the message-sending pattern of activity between the CD and Mkt Info classes, Anne suggested that they should be in the same partition. Furthermore, since the Vendor class was only related to the CD class, Alec suggested that they be placed in the same partition. Finally, the development team decided to place the Order and Order Item classes together and they decided that the "temporary" classes used during the shopping process should be grouped together: Search Req, CD List, Shopping Cart, and SC Item classes together in their own partitions. If you remember, in the previous installation, Anne had recommended including a Shopping Cart class. Once they did that, they realized that they also needed to include a Shopping Cart Item class.

The third step was to model each of these partitions as packages. Figure 7-A shows the classes being contained in their respective packages. Observe that the Credit-Card center currently is not contained in any package.

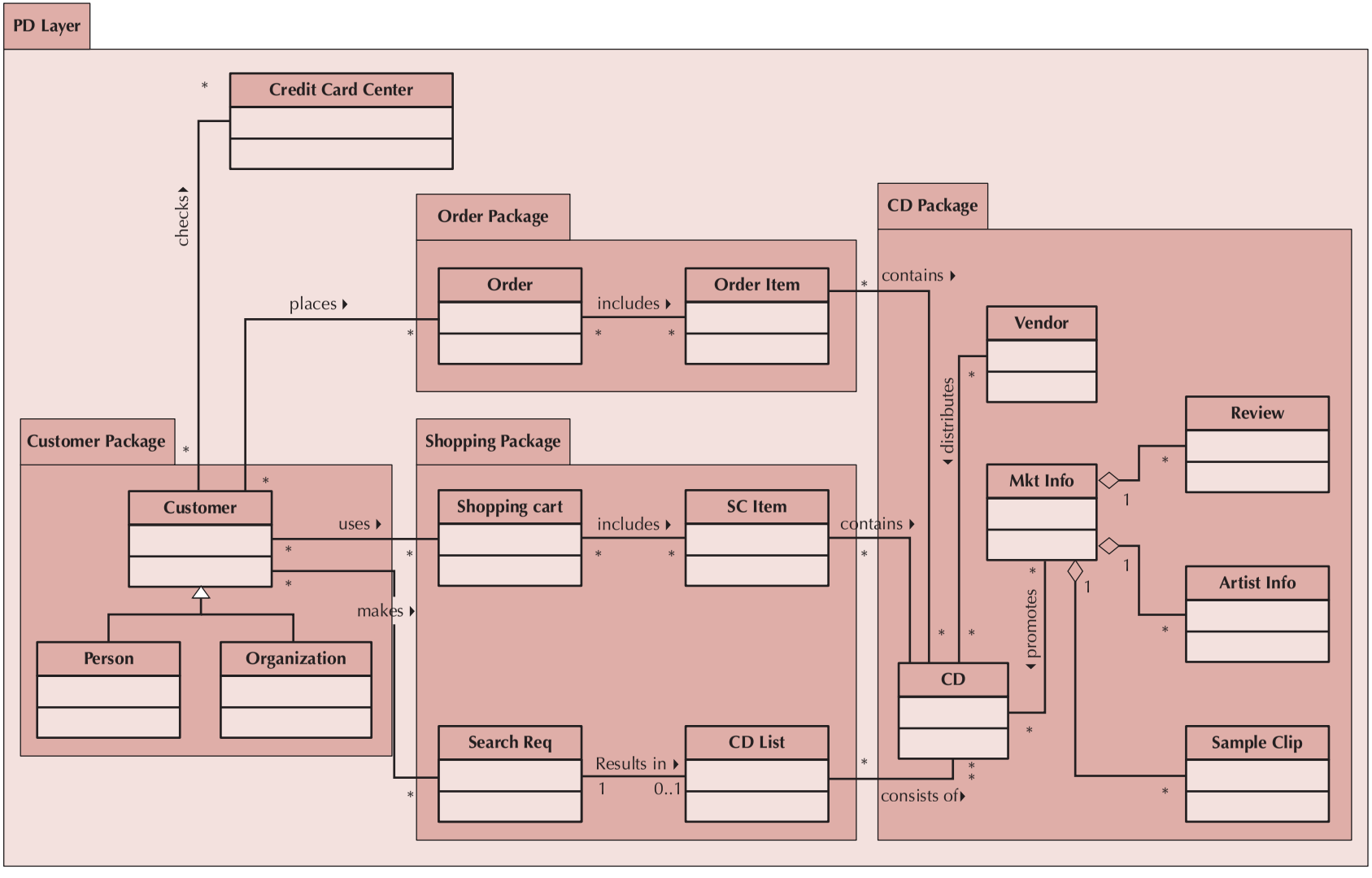

FIGURE 7-A Package Diagram of the PD Layer of CD Selections Internet Sales System

Next, Alec quickly identified four associations among the different packages: the Customer Package and the Order Package, the Customer Package and the Shopping Package, the Order Package and the CD Package, and the Shopping Package and the CD Package. He also identified an association between the Credit Card Center class and the Customer Package. Based on these associations, five dependency relationships were identified.

The fifth and final step was to place the dependency relationships on the package diagram. Again, to increase the understandability of the dependency relationships among the different packages, Alec decided to create a pure package diagram that only depicted the highest-level packages (and in this case the Credit Card Center class) and the dependency relationships (see Figure 7-B).

Verifying and Validating the Analysis Models

Upon completing the partitioning of the Problem Domain Layer, the team felt pretty good about what they had accomplished. However, based on his understanding of what was coming up next, Alec wanted to be sure that the analysis models–functional, structural, behavioral–and the partitions made sense, he decided that all of the work to date needed to go through a verification and validation step. To say the least, the team was not all that excited about this. In fact, Brian pointed out that the team had been verifying and validating everything as they went along. As such, he argued that this would

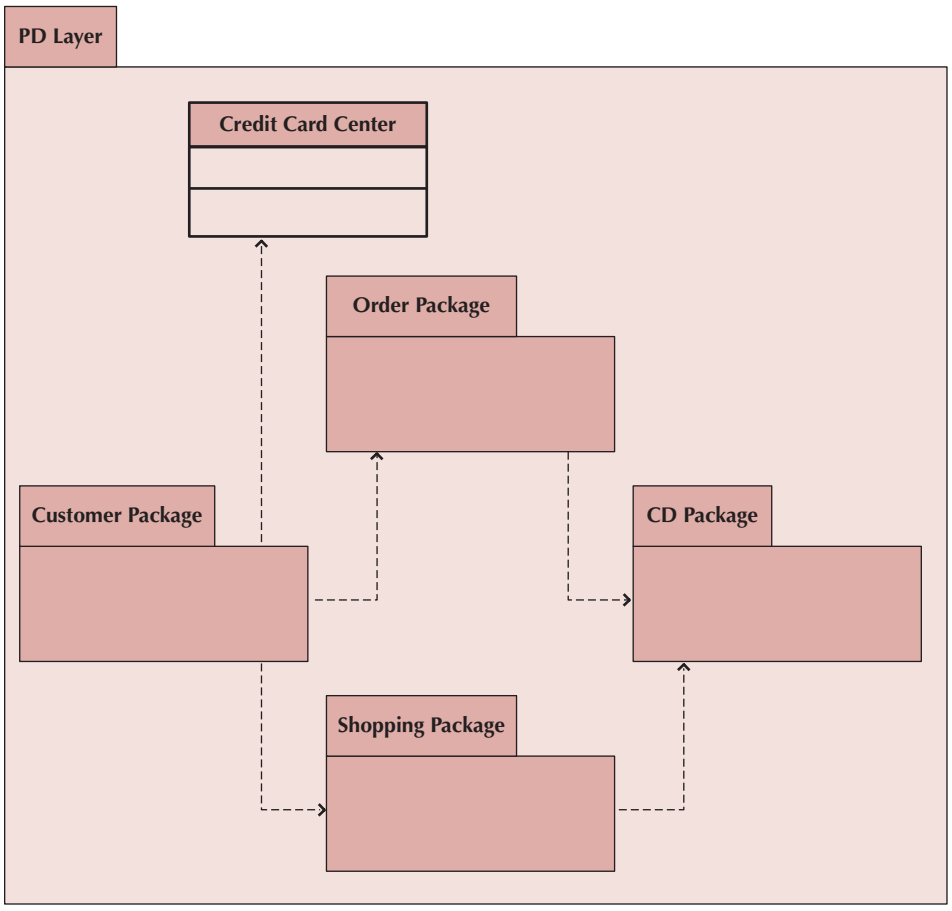

FIGURE 7-B
Overview Package Diagram of the PD Layer of CD Selections Internet Sales System

simply be a waste of time. But, Alec prevailed. He explained that in past projects, when they had not assured the quality of the Problem Domain Layer that teams had run into significant problems. These problems included the system not solving the right problem, significant cost overruns, and the system not being delivered on time. Since, this team was relatively inexperienced with the technology that they were about to use, Alec told the team, that there was not enough slack in the workplan to run into problems related to the analysis models. He suggested that the team perform a walkthrough with the analysis models and to ensure that all the relationships among the diagrams were fully tested (see Figures 7-1, 7-2, 7-8, 7-13, 7-16)).

The good news was that given all of the verification and validation that they had performed on the individual models, the team did not uncover any additional errors within the models. Brian sort of brought up his earlier point in a "I told you so" manner. However, Alec let him blow off a little steam and simply reminded the team that it was better to have done the verification and validation step now and not have to be sorry later. He pointed out that the other layers are mostly dependent on the Problem Domain Layer (see Figure 7-19) and any mistake not caught now could be very costly to catch it later.

Developing the Actual Design

Now that the team had verified and validated the analysis models, Alec had to decide on a design strategy. As he saw it, he had three different approaches that he could take with the new system: he could develop the entire system using development resources from CD Selections, he could buy a commercial Internet sales packaged software program (or a set of different packages and integrate them), or he could hire a consulting firm or service provider to create the system. Immediately, Alec ruled out the third option. Building Internet applications, especially sales systems, was important to the CD Selections' business strategy. By outsourcing the Internet Sales System, CD Selections would not develop Internet application development skills and business skills within the organization.

Instead, Alec decided that a custom development project using the company's standard Web development tools would be the best choice for CD Selections. In this way, the company would be developing critical technical and business skills in-house, and the project team would be able to have a high level of flexibility and control over the final product. Also, Alec wanted the new Internet Sales System to directly interface with the existing distribution system, and there was a chance that a packaged solution would not be able to integrate as well into the CD Selections environment.

There was one part of the project that potentially could be handled using packaged software: the shopping cart portion of the application. Alec realized that a multitude of programs have been written and are available (at low prices) to handle a customer's order transaction over the Web. These programs allow customers to select items for an order form, input credit card and billing information, and finalize the order transaction. Alec believed that the project team should at least consider some of these packaged alternatives so that less time had to be spent writing a program that handled basic Web tasks, and more time could be devoted to innovative marketing ideas and custom interfaces with the distribution system.

To help better understand some of the shopping cart programs that were available in the market and how their adoption could benefit the project, Alec created an alternative matrix that compared three different shopping cart programs to one another (see Figure 7-C). Although all three alternatives had positive points, Alec saw Alternative B (WebShop) as the best alternative for handling the shopping cart functionality for the new Internet sales

	Alternative 1: Shop-With-Me	Alternative 2: WebShop	Alternative 3: Shop-N-Go
Technical Feasibility	• Developed using C: very little C experience in-house • Orders sent to company using email files	• Developed using C and JAVA: would like to develop in-house JAVA skills • Flexible export features for passing order information to other systems	• Developed using JAVA: would like to develop in-house JAVA skills • Orders saved to a number of file formats
Economic Feasibility	• $150 initial charge	• $700 up front charge, no yearly fees	• $200/year
Organizational Feasibility	• Program used by other retail music companies	• Program used by other retail music companies	• Brand new application: few companies have experience with Shop-N-Go to date
Other Benefits	• Very simple to use	• Tom in IS support has had limited, but positive experience with this program • Easy to customize	
Other Limitations			• The interface is not easily customized

FIGURE 7-C Alternative Matrix for Shopping Cart Program

system. WebShop was written in JAVA, the tool that CD Selections selected as its standard Web development language; the expense was reasonable, with no hidden or recurring costs; and there was a person in-house who had some positive experience with the program. Alec made a note to look into acquiring WebShop as the shopping cart program for the Internet sales system.