

Chickens, Eggs, and Use Cases

It's not a question of which comes first, but rather how to put it all together.

This month I'll examine a question that I'm asked repeatedly: "Which should be created first: the class model or the use cases?"

To most OO analysts and developers, the answer is obvious: start with use cases. After all, that's what the popular OO books and development methodologies say to do. OO Orthodoxy says that by starting with use cases, it becomes easier to find classes, attributes, and associations. Some texts assert that the use cases themselves lead directly to classes. Even my own book puts the use case chapter (Chapter 4) before the chapters on class modeling (Chapters 5 and 6).

Yet in a prior column, I described starting analysis with an "object blitz" and, in practice, often sketch a class diagram before beginning to talk about use cases and operational scenarios. Is there a rational explanation, or am I just being contrarian?

Dueling Perspectives

Use cases look at the problem from a "who can do what things" perspective. To do a use-case-based analysis of an online bookstore we'd look for different actors—people or things that interact with the system—and the use cases—distinct user-observable behaviors that must be exhibited by the system. In UML, these are represented diagrammatically with actors that represent different user roles showing who will initiate or otherwise participate in a use case. An initial brainstorming of actors and use cases might result in a set such as that in Figure 1.

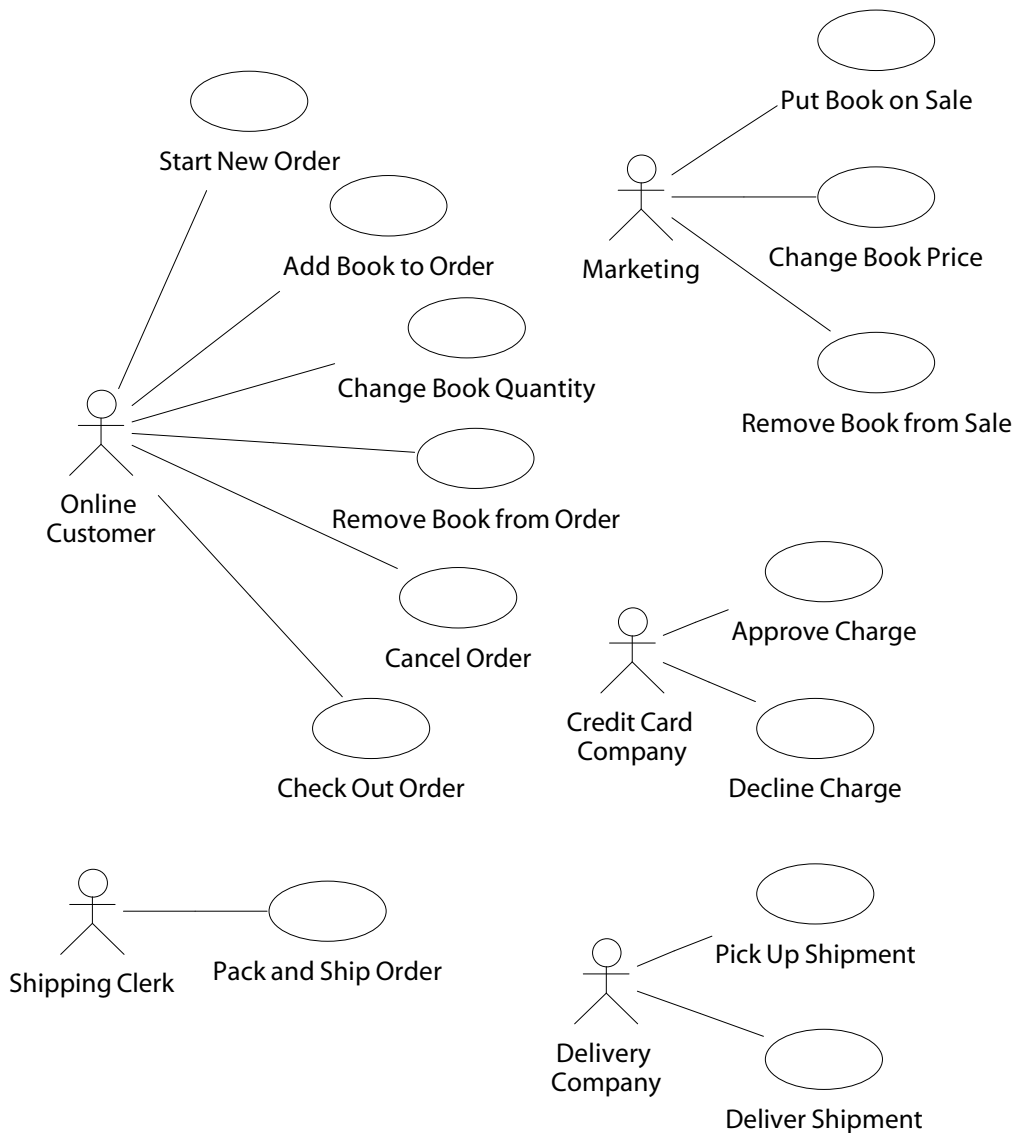


Figure 1: Online Bookstore Use Cases

Classes look at the problem from a “what are the things” perspective. A class model, expressed as a UML class diagram, is a static statement of the things in the problem, expressed in terms of classes, attributes, and associations. An object blitz of the bookstore might identify classes such as book, author, publisher, order, charge, shipment, and such. Those classes are related to one another as shown in the class diagrams:

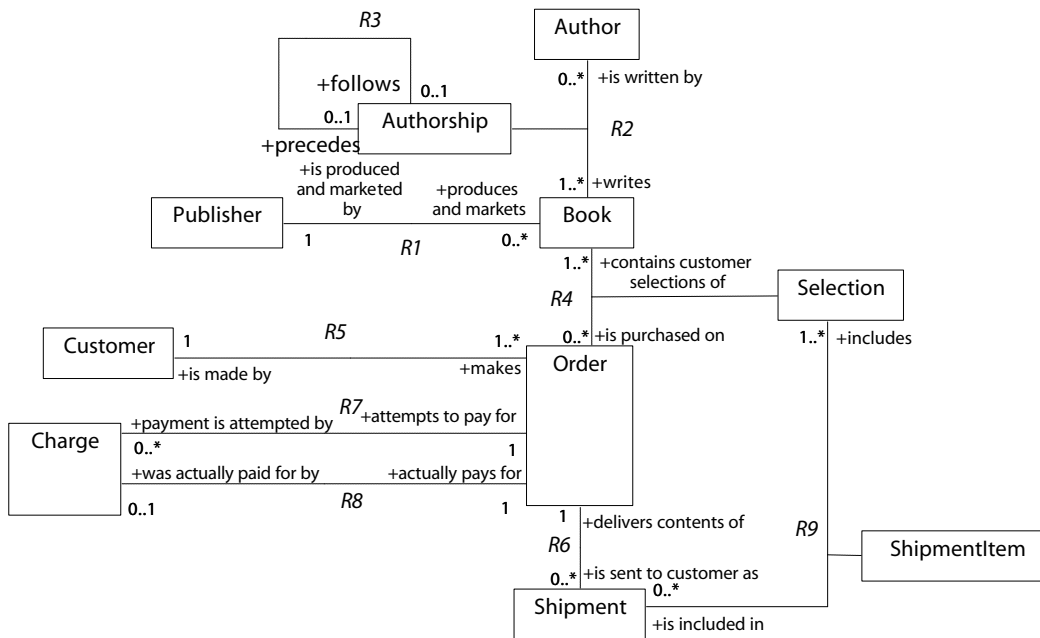


Figure 2: Online Bookstore Classes

I'm immediately struck by the commonality between the two pictures. Of course, there are some things in one picture that are not in the other—but that's to be expected since these are initial brainstormings from different perspectives. It's not a question of which perspective is more right or more appropriate—instead, how can we use the two perspectives to create a complete, whole picture of a domain?

Just Start

Because use cases and class models approach a domain from different but complementary perspectives, they can be useful in different settings and different kinds of problems. There's no sense in arguing how to start—just start.

If the problem is posed from a users-do-things perspective, it may be easier to first blitz use cases. On the other hand, if a problem is initially expressed in terms of data (e.g. as a collection of forms and screens and tables), it may be easier to start with the class model. One perspective is not more right than the other. Rather, strive for a goal to express the problem from both perspectives. Identify classes—and identify use cases. Identify use cases—and identify classes.

Use Cases to Classes

The relationship between use cases and classes is not one-to-one, however. Use cases do not immediately lead to good classes in the sense that each use case becomes a class. (Some approaches advocate creating “controller” classes for each use case, but these

single instance controller classes are not a very effective way to model a domain.) Instead, use the vocabulary in the use cases to identify classes. These can come from

- the actors: Customer, Shipping Clerk, Marketing Rep, Credit Card Company, Delivery Company
- the subject nouns: Order, Book, Shipment, Charge
- some process names: Check Out, Pick Up, Approve, Decline (However, be careful not to create the dreaded single-instance controller classes.)

While class identification is more than identifying the nouns, this can be a good start.

Classes to Use Cases

Now if you've started with a class model and need to define use cases, a good place to start is with the basic CRUD—create, read, update, and delete—for each class.

Class	Created when	Updated when	Deleted when
Customer	Customer checks out an order and new customer info is specified	order is checked out and an existing customer's info is changed	Never
Book	Marketing puts a book on sale	Book price is changed or a book is taken off sale	Never
Order	Customer starts an order	Items are added to or removed from an order, order is checked out	Order not yet checked out is canceled
Charge	Customer checks out an order or resubmits a charge	Credit card company approves or declines a charge	Never
Shipment	Shipping clerk packs an order for shipment	Delivery company picks up and later delivers an order	Never

You can also create a similar table for each association.

By now, you may notice that there is an iterative process at work here. The vocabulary of the use cases may identify additional classes, and the simple lifecycles of the classes may identify additional use cases. For example, the idea of resubmitting a charge does not appear in the original use case list, but becomes apparent as a possibility when we look at the question of when Charge objects are created, and note the multiplicity of the association between an Order and a Charge.

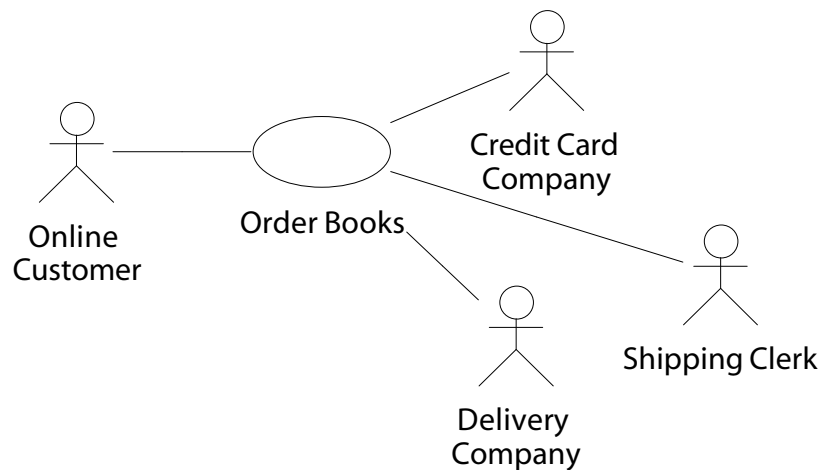
So the answer is...?

Use cases and class models are both appropriate starting points in an object-oriented analysis. How to start is often a function of the nature of the problem and the skills of the modeler. Each contributes a different and complementary perspective to a solution, and an iterative approach in which a modeler creates both concurrently can create a better analysis than could be created by focusing on only one model at a time.

Use Cases “On the Level”

In writing this column, I received feedback from several reviewers who said, “of course, these use cases look strikingly like your class model...because you probably started from the class model.”

“OK, then,” I asked, “what would your use cases have looked like?” I got back a picture that looked like this:



It’s one big use case bubble with many actors. Yes, the use case covers the entire process of placing an order and receiving books. But it’s significantly less detailed. Is it more right or less right than the use cases I’ve drawn in Figure 1? It’s not more or less right—it’s merely at a different level of detail.

Each of the use cases in Figure 1 is initiated by a single actor—I call them “single system interaction” use cases because once the actor initiates the use case, no other actor’s intervention is required to complete it. It’s a nice detailed view, but it lacks context. The use case in Figure 3 provides that context: many steps, initiated by different actors, are required to order books. How can we connect the two use case views?

In *Executable UML*, Steve and I use UML activity diagrams with swimlanes to connect the two levels of use cases. Figure 4 shows the big Order Books use case as a sequence of single-system-interaction use cases.

