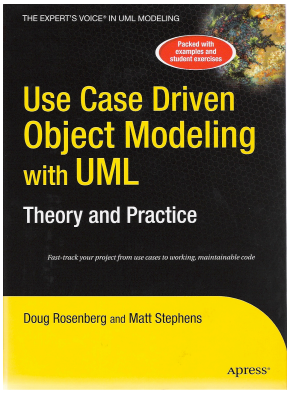
Domain Modelling Example

This example is from the book Use Case Driven Object Modeling with UML, by Doug Rosenburg and Matt Stephens, Apress, 2007, 1-59059-774-5



Internet Bookstore: Extracting the First-Pass Domain Model from High-Level Requirements

When you're creating your domain model, a good source of domain classes includes the high-level requirements—the ones that are usually (but not always) written in the form "The system *shall* do this; the system *shall not* do that." It's useful to scan these requirements, extracting the nouns and noun phrases. You can then refine these to create the initial domain model.

With that in mind, let's go through the high-level requirements for the Internet Bookstore and extract some **domain classes** from them.

1. The **bookstore** will be web based initially, but it must have a sufficiently flexible architecture that alternative front-ends may be developed (Swing/applets, web services, etc.).
2. The bookstore must be able to sell **books**, with **orders** accepted over the **Internet**.
3. The user must be able to add books into an online **shopping cart**, prior to **checkout**.
    a. Similarly, the user must be able to remove **items** from the shopping cart.
4. The user must be able to maintain **wish lists** of books that he or she wants to purchase later.
5. The user must be able to cancel orders before they've shipped.
6. The user must be able to pay by **credit card** or **purchase order**.
7. It must be possible for the user to return books.
8. The bookstore must be embeddable into **associate partners**' websites using **mini-catalogs**, which are derived from an overall **master catalog** stored in a central **database**.
    a. The mini-catalogs must be defined in XML, as they will be transferred between this and (later to be defined) external systems.
    b. The **shipping fulfillment system** shall be carried out via Amazon Web Services.
9. The user must be able to create a **customer account**, so that the system remembers the user's details (name, address, credit card details) at login.
    a. The system shall maintain a **list of accounts** in its central database.
    b. When a user logs in, his or her **password** must always be matched against the passwords in the **master account list**.
10. The user must be able to search for books by various **search methods—title**, **author**, **keyword**, or **category**—and then view the **books' details**.
11. It must be possible for the user to post reviews of favorite books; the **review comments** should appear on the book details screen. The review should include a **customer rating** (1–5), which is usually shown along with the book title in **book lists**.

   a. **Book reviews** must be moderated—that is, checked and "OK'd" by a member of staff before they're published on the website.

   b. Longer reviews should be truncated on the book details screen; the **customer** may click to view the full review on a separate page.

12. It must be possible for staff to post **editorial reviews** of books. These should also appear on the book details screen.

13. The bookstore shall allow third-party **sellers** (e.g., second-hand bookstores) to add their own individual **book catalogs**. These are added into the overall **master book catalog** so that sellers' books are included in search results.

14. The bookstore must be scalable, with the following specific requirements:

   a. The bookstore must be capable of maintaining **user accounts** for up to 100,000 customers in its first six months, and then a further 1,000,000 after that.

   b. The bookstore must be capable of serving up to 1,000 simultaneous users (10,000 after six months).

   c. The bookstore must be able to accommodate up to 100 search requests per minute (1,000/minute after six months).

   d. The bookstore must be able to accommodate up to 100 purchases per hour (1,000/hour after six months).

These requirements are a rich source of domain classes. Let's put all the highlighted nouns and noun phrases into a list (in the process, we'll turn all the plurals into singulars, and put them all in alphabetical order):

Associate Partner
Author
Book
Book Catalog
Book Details
Book List
Book Review
Bookstore
Category
Checkout
Credit Card
Customer
Customer Account
Customer Rating
Database
Editorial Review
Internet
Item
Keyword
List of Accounts
Master Account List
Master Book Catalog
Master Catalog
Mini-Catalog
Order
Password
Purchase Order
Review Comment
Search Method
Search Results
Seller
Shipping Fulfillment System
Shopping Cart
Title
User Account
Wish List

There's quite a bit of duplication in this list; similar terms are being used for basically the same thing. But that's really the main benefit of the domain modeling approach: you get to identify and eliminate these duplicate terms early on in the project.

■ **Exercise Disambiguation via Grammatical Inspection**: We'll go through this list next, whipping it into shape and eliminating the duplicate terms. But first, try to identify the six duplicate pairs in the list. (Be careful: one pair *seems* like a duplicate but really isn't.)

Some of the items in the list are simply unnecessary because they fall outside the scope of the domain model, or they're actions sneakily masquerading as nouns.

Let's step through the list now and tune it up a bit:

- You'd *think* that the terms "Customer" and "Customer Account" are duplicates, but in fact they represent subtly different things: "Customer Account" is an entity stored in the database, whereas "Customer" is an actor (see the next item in this list).
- "Customer" and "Seller" are actors, and thus should be placed on use case diagrams. (See Chapter 3.)
- The terms "User Account" and "Customer Account" are duplicates. The choice of which one to keep is fairly arbitrary, so we'll go with "Customer Account."
- The terms "List of Accounts" and "Master Account List" are duplicates, so one of them should be removed. As we also have a "Master Book Catalog," the consistent thing would be to keep "Master Account List."
- The terms "Book Review" and "Review Comment" are duplicates, so we'll keep "Book Review."
- We have several different candidate terms for a catalog, or list of books: "Book Catalog," "Book List," "Mini-Catalog," and "Master Catalog." Catalogs and lists are probably different concepts. In fact, it seems that the requirements are trying to tell us something, which may just be implied in the text. When in doubt, *talk to the customer.* Ask questions until you get a clear, unambiguous answer.

  "Book Catalog" and "Master Catalog" are in fact the duplicates here, so we'll keep "Master Catalog," as it provides a good contrast with "Mini-Catalog." "Book List," meanwhile, is probably an umbrella term for different types of lists; we'll keep it in there for now and see how it fits in when we draw the domain model diagram.
- There's another duplicate in this area: "Master Catalog" and "Master Book Catalog." We'll delete "Master Catalog," as "Master Book Catalog" is the more descriptive term.
- The word "Internet" is too generic and doesn't add anything here.
- The word "Password" is a too small to be an object and would be shown as a UI element, so we should remove it from the domain model. If we start to include all the UI elements in the domain model, we're opening a serious can of worms and could be here all night backed into a corner, fighting them away with a large stick.

- Same goes for "Title" and "Keyword."
- Yet another duplicate is "Book" and "Book Details." We'll just keep "Book," as it's more concise than "Book Details," without losing any meaning.
- The word "Item" is just vague and fuzzy, but it does represent a valid concept: an item that's been added to the user's shopping cart. So we'll rename it "Line Item" and keep it in the list.
- The word "Bookstore" is a bit too broad and is unlikely to be referred to explicitly, so we can get rid of it.

Following is the updated list of candidate domain classes. Figure 2-4 shows those classes laid out in a class diagram.

Associate Partner	Customer Account	Order
Author	Customer Rating	Purchase Order
Book	Database	Search Method
Book List	Editorial Review	Search Results
Book Review	Line Item	Shipping Fulfillment System
Category	Master Account List	Shopping Cart
Checkout	Master Book Catalog	Wish List
Credit Card	Mini-Catalog	

As we mentioned earlier, although grammatical inspection techniques are useful to get a quick start, you shouldn't spend weeks or even days doing this. (As you'll see in Chapter 4, the rest of the objects for the Internet Bookstore were identified during robustness analysis.) A couple of hours is about the right amount of time to spend on the domain model before getting started writing the use cases.

■ **Caution** Don't get bogged down in grammatical inspection.

Figure 2-4 shows one type of relationship, *aggregation* (aka has-a), which we described earlier. As this is a first-pass attempt, not all the relationships shown are correct.

A helpful technique is to read the diagram aloud and include the term "has-a." For example, a Shopping Cart "has" Line Items. But, does an Order "have" Checkouts? Perhaps not. Notice that a few of the domain objects currently don't match up with anything else (namely, Associate Partner, Shipping Fulfillment System, Database, and Search Method). We've grouped these together over on the right for now; during robustness analysis, these may get linked to other objects, warped into something different, or removed altogether.

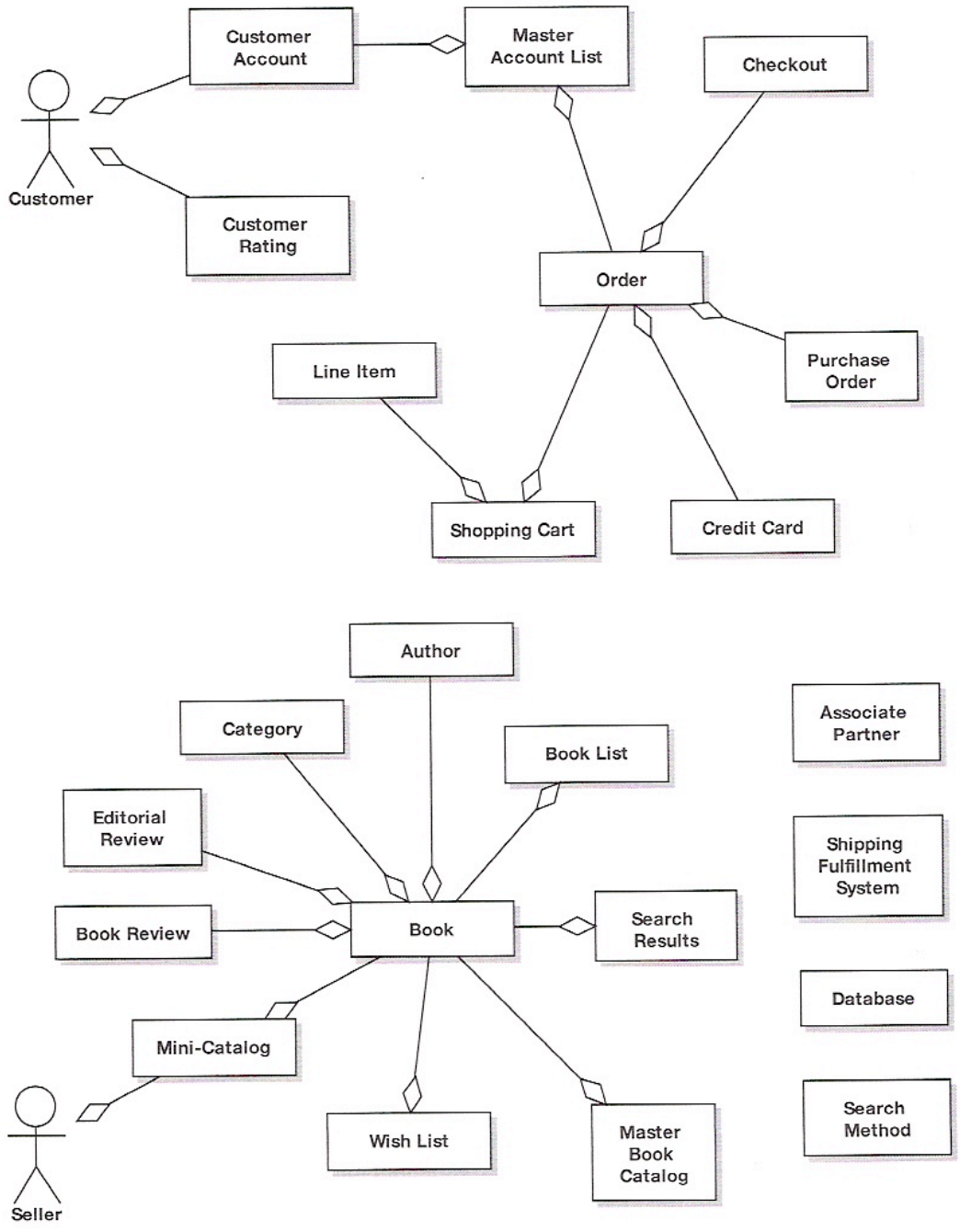


Figure 2-4. *First-pass domain model for the Internet Bookstore project*

There's still some work that needs to be done on this domain model before we're ready to move on to the next stage, so let's do some more tidying up work next. Hopefully, this will help to illustrate an important element of the ICONIX approach: ***continuous improvement via ongoing iteration and refinement***.

Internet Bookstore: Second Attempt at the Domain Model

When drawing up the domain model diagram, you're generally brainstorming as a team. Often the team will identify further domain objects that weren't in the requirements, but instead have been dredged from somebody's own understanding of the problem domain. To illustrate this, let's say we've discovered two additional domain objects: `Order History` and `Order Dispatch`. These weren't mentioned explicitly in the requirements, but they could still classify as minimum requirements for an Internet bookstore.

The updated diagram is shown in Figure 2-5, with the new domain classes shown in red.

In Figure 2-5, we've explored the concept of order fulfillment and dispatch. `Shipping Fulfillment System` still remains on the diagram, but we'll have to decide whether this is in scope for the current model or it's an external system that we need to interface with. **External systems are always modeled as actors.**

We've also removed `Checkout`, as on reflection this was really a verb in noun's clothing. And we've removed `Author`, as this is really just another field in the `Book` (i.e., it's too small to be a first-class object on the domain model[1]). Authors . . . who needs 'em?

There's some ambiguity around `Master Book Catalog`, which we've attempted to resolve. We've removed the link between `Book` and `Master Book Catalog`, and instead added a class called `Book Catalog` and linked `Book` to that instead. So we end up with a **tangle of relationships**: we're effectively saying that a `Book` belongs to a `Book Catalog`, and a `Book Catalog` belongs to a `Master Book Catalog` (i.e., a `Master Book Catalog` is really a catalog of catalogs). Ideally, a `Mini-Catalog` should also belong to the `Master Book Catalog`. But this tangle is getting complicated. What we really need is a simple way of saying that a `Book` can belong to `Book Catalogs`, and there can be various *types* of `Book Catalogs`. Luckily, a light sprinkling of ***generalization*** can work wonders on such relationship tangles, as you'll see in the next section.

1. Remember we're not creating a data model here. If this were a database design, we would almost certainly create a separate Authors table.

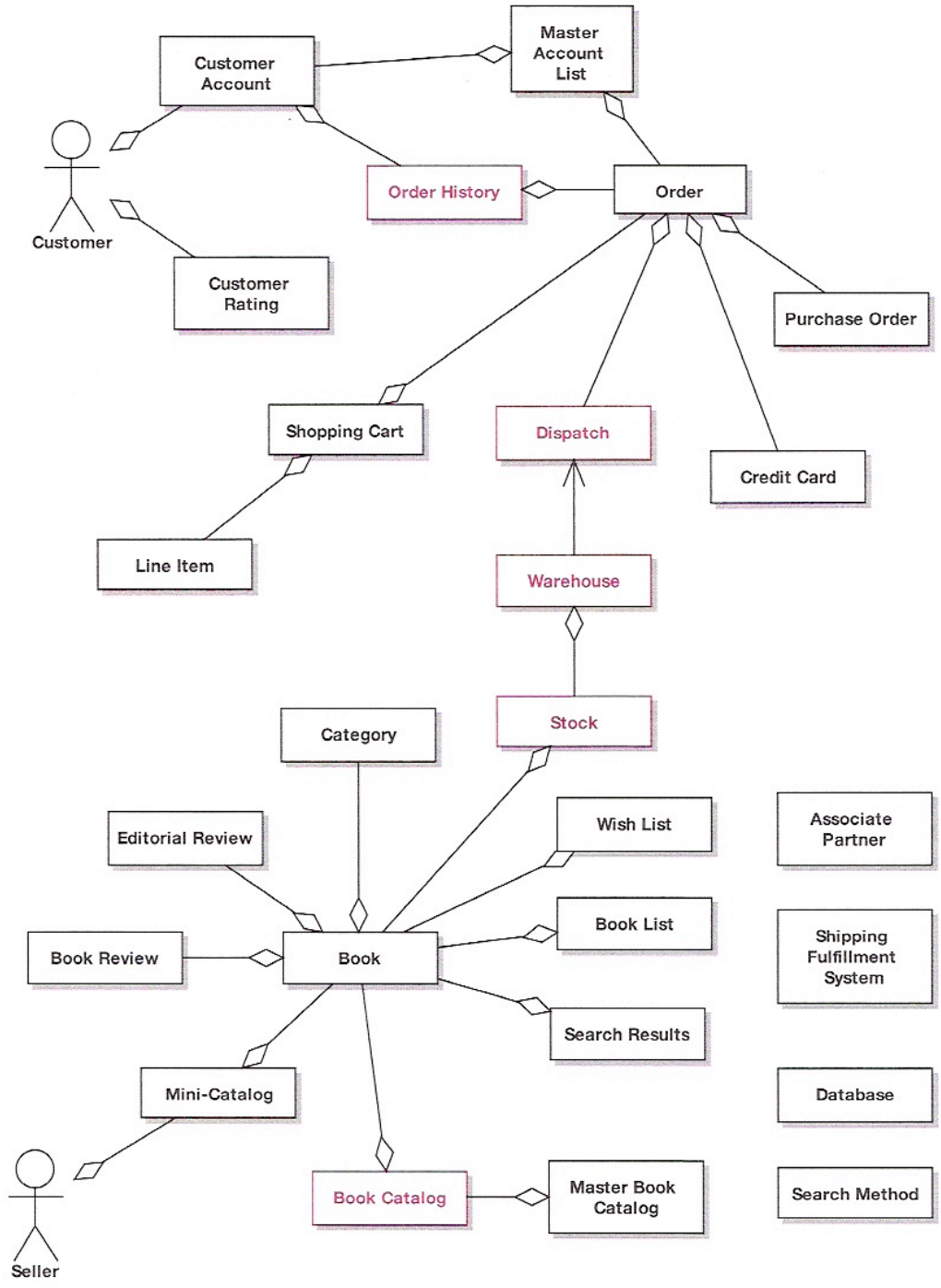


Figure 2-5. *Second snapshot of the evolving domain model for the Internet Bookstore project*

Internet Bookstore: Building Generalization Relationships

A *generalization relationship* is one in which one class is a "kind of" some other class—for example, a `Cat` is a kind of `Animal`. This is why generalization is often called an *is-a* relationship.

`Cat` is more specific than `Animal` (`Cat` is a "refinement" of the more general `Animal` class), hence the term "generalization." The more specific class is called the *subclass*, and the more general class is the *superclass*. Creating subclasses of more general classes is known as *subtyping*.

Within the Internet Bookstore, `Book Catalog` is a good candidate for subtyping, because doing so will help to "de-cloud" the relationship between `Mini-Catalog` and `Master Book Catalog`. `Book List` is also a good candidate for subtyping, because there may well be different types of accounts and different types of book lists.

As we delve more deeply into the user's needs for the Internet Bookstore system, we're beginning to identify different types of book lists: customer wish lists, recommendation lists, Related Books, Search Results, and so on. It's becoming clear that these are all simply lists of `Books`, so they could (conceptually, at least) have a common parent class. We've discovered that there are indeed aspects of `Wish Lists`, `Related Books`, and so on that are different enough to justify separate treatment, while they still have enough in common that they're all kinds of `Book List`. Figure 2-6 shows the notation for this generalization structure.

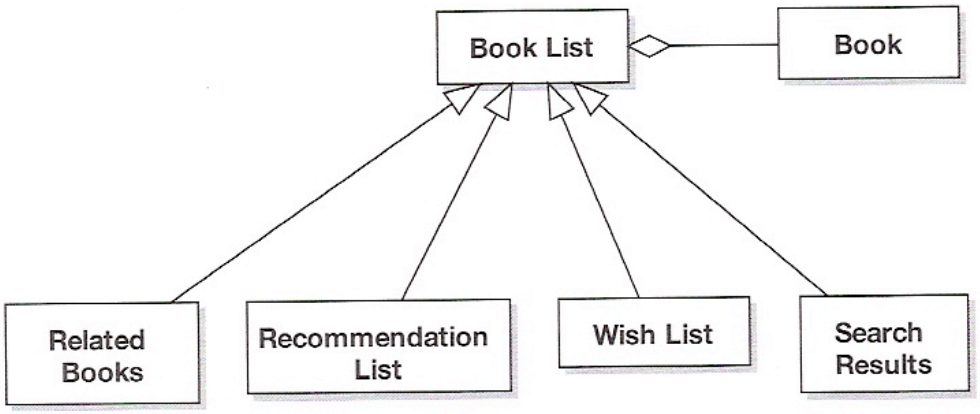


Figure 2-6. *Book Lists detail from the Internet Bookstore domain model*

The new classes (`Related Books`, `Recommendation List`, `Wish List`, and `Search Results`) inherit the attributes and operations that we define for `Book List`. Let's read this diagram out loud: A book list has books. Related Books is a book list. Recommendation List is a book list. Wish List is a book list. Search Results is a book list. All true statements that describe the problem space? Great, let's move on.

■ **Tip** You could also add additional specialized attributes and operations for each of the new classes. In other words, if you were to add an operation to `Related Books`, it would only be available to `Related Books`. However, if you add it to `Book List`, the new operation would be available to all of its subclasses.

Figure 2-7 shows the updated Internet Bookstore domain model, which makes good use of generalization to clarify the relationships between the domain classes. The new classes are shown in red.

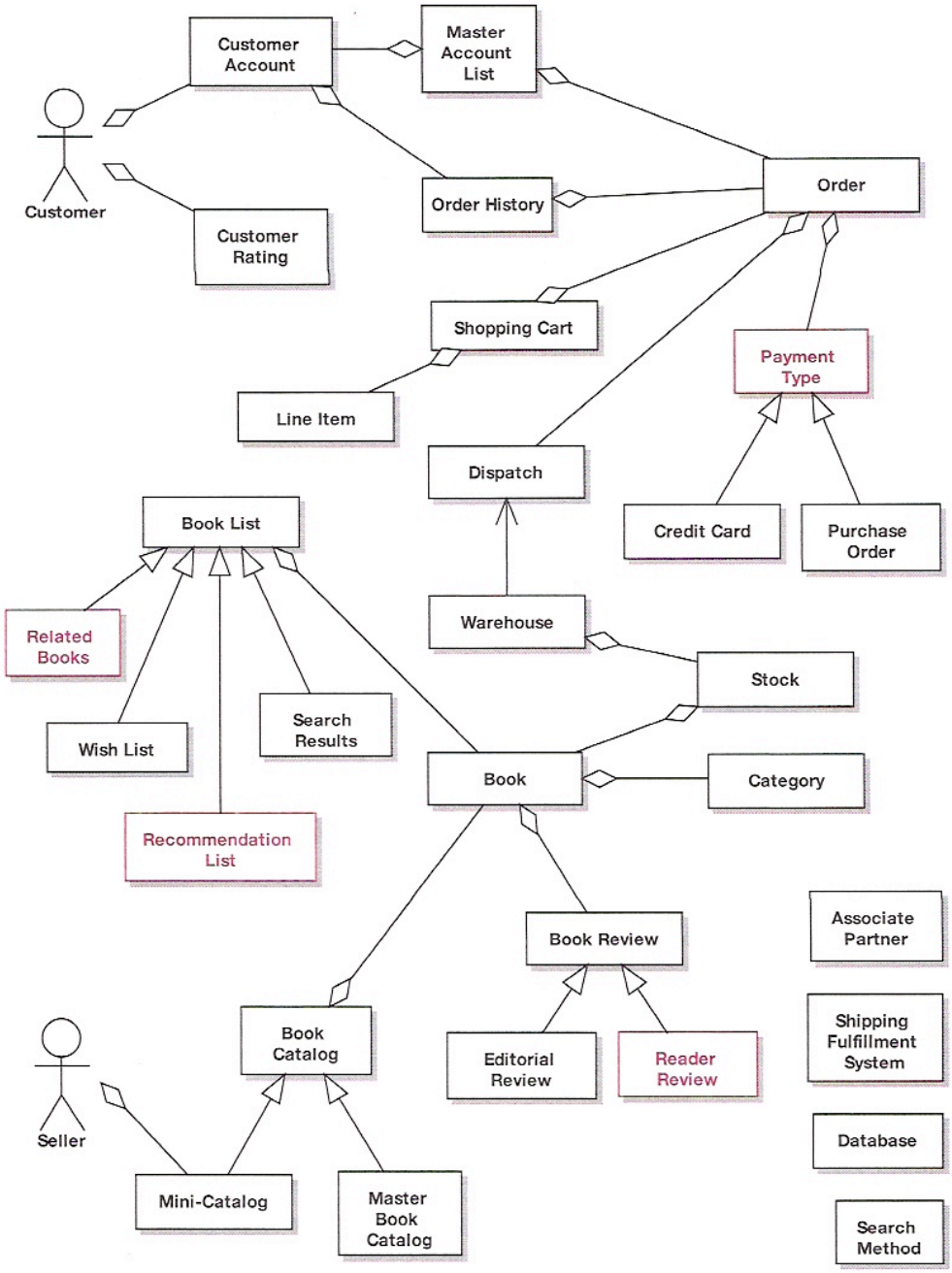


Figure 2-7. *Third snapshot of the evolving domain model for the Internet Bookstore project*