.5 Understanding the System Context Using a Domain Model

6.5.1 What Is a Domain Model?

A domain model captures the most important types of objects in the context of the system. The domain objects represent the "things" that exist or events that transpire in environment in which the system works [2, 5].

Many of the domain objects or classes (to use more precise terminology) can be found from a requirements specification or by interviewing domain experts. The domain classes come in three typical shapes:

- Business objects that represent things that are manipulated in a business, such as orders, accounts, and contracts.
- Real-world objects and concepts that a system needs to keep track of, such as enemy aircraft, missiles, and trajectory.
- Events that will or have transpired, such as aircraft arrival, aircraft departure, and lunch break.

The domain model is described in UML diagrams (particularly in class diagrams).

These diagrams illustrate to customers, users, reviewers, and other developers the domain classes and how they are related to one another by association.

Example **The Domain Classes Order, Invoice, Item, and Account**

The system will use the Internet to send orders, invoices, and payments between buyers and sellers. The system helps the buyer prepare orders, the seller to evaluate orders and send invoices, and the buyer to validate invoices and effect payment from the buyer's account to that of the seller.

An order is the request from a buyer to a seller for a number of items. Each item "occupies a line" in the order. An order has attributes such as date of submission and delivery address. See the class diagram in Figure 6.3.

An invoice is a request for payment sent from a seller to a buyer in response to an order for goods or services. An invoice has attributes such as amount, date of submission, and last date of payment. An invoice may be the request for payment of several orders.

An invoice is paid by transferring money from the buyer's account to the seller's account. An Account has attributes such as balance and owner. The attribute owner identifies the person who owns the account.

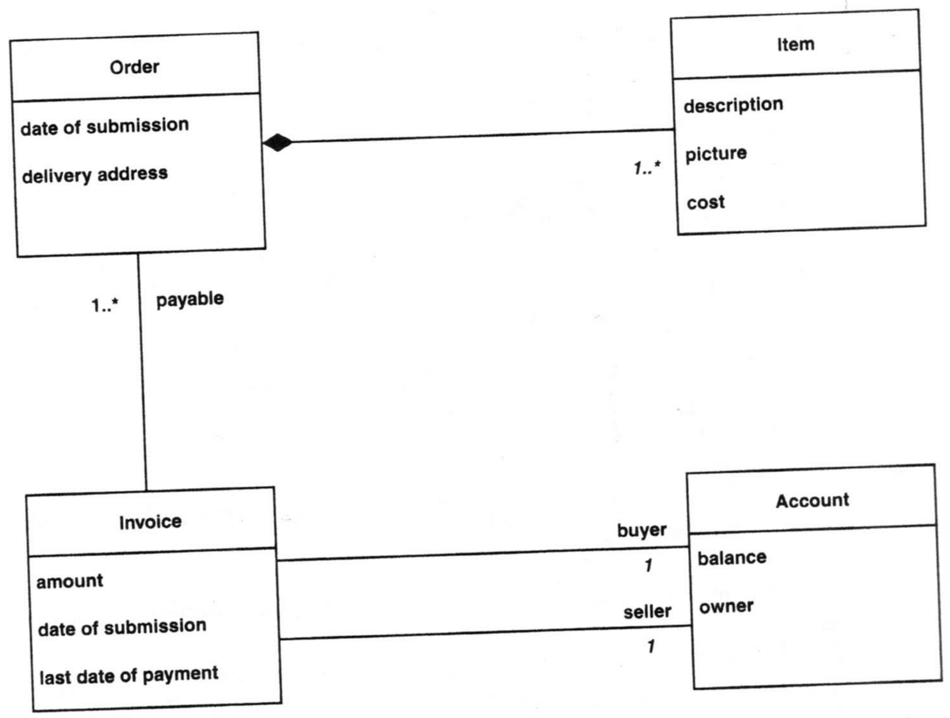

FIGURE 6.3 A class diagram in a domain model, capturing the most important concepts in the context of the system.

UML Notation

Classes (rectangles), attributes (text in the lower half of the class rectangles), and associations (the lines between the class rectangles). The text at the end of an association path explains the role of one class in relation to another, that is, the role of the association. The multiplicity—the numbers and stars at the end of an association path—tells how many objects of the class at this end are linked to one object at the other end. For example, the association connecting the classes Invoice and Order in Figure 6.3 has a 1..* multiplicity adorned to the Order class end. This means that each Invoice object may be a request for payment of one or more Order objects, as indicated by the payable association **role** (Appendix A).

6.5.2 Developing a Domain Model

Domain modeling is usually done in workshops by domain analysts, who use UML and other modeling languages to document the results. To form an effective team, these workshops should include both domain experts and people who are skilled in modeling.

The purpose of domain modeling is to understand and describe the most important classes within the context of the domain. Modest-sized domains usually require between 10 and 50 such classes. Larger domains may require many more.

The remaining hundreds of candidate classes that the analysts may elicit for the domain are kept as definitions in a glossary of terms; otherwise, the domain model will become too large and will require more effort than is appropriate for this stage of the process.

Sometimes, such as for very small business domains, it is not necessary to develop an object model for the domain; instead, a glossary of terms may suffice.

The glossary and domain model help users, customers, developers, and other stakeholders use a common vocabulary. Common terminology is necessary to share knowledge with others. Where confusion abounds, engineering is difficult, if not impossible. And to build a software system of any size, modern engineers must "merge" the language of all the participants into a consistent one.

Finally, a word of caution regarding domain modeling is in order. It can be very easy to start modeling the internal parts of a system and not its context [7]. For example, some domain objects might have a straightforward representation in the system, and some domain analysts might in turn fall into the trap of specifying details regarding this representation. In such cases it is very important to keep in mind that the purpose of domain modeling is to contribute to an understanding of the system's context, and thereby also to an understanding of the system's requirements as they originate from this context. In other words, domain modeling should contribute to an understanding of the *problem* that the system is supposed to solve in relation to its context. The system's internal way of solving this problem will be dealt with in the analysis, design, and implementation workflows (see Chapters 8, 9, and 10).

6.5.3 Use of the Domain Model

The domain classes and the glossary of terms are used when developing the use case and analysis models. They are used

- When describing the use cases and when designing the user interface, something which we will return to in Chapter 7.
- To suggest classes internal to the developed system during analysis, something which we will return to in Chapter 8.

However, there is an even more systematic way to identify use cases and to find classes inside the system: develop a business model. As we will see, a domain model