Chapter 4 – Domain Modeling

Table of Contents
- Chapter Overview
- Learning Objectives
- Notes on Opening Case and EOC Cases
- Key Terms

Chapter Overview
The focus of this chapter is to identify and define the entities or problem domain classes. Traditional system development methodologies called these “things” data entities and used Entity-Relationship diagrams to model the data structure. Object-oriented techniques call these “things” problem domain classes, or just classes for short. A UML class diagram is used to model the data structure. Either type of model can be used and either results in equivalent solutions. You should be familiar with both Entity-Relationship diagrams and Class diagrams.

The first part of the chapter is a discussion to help you find, identify, and classify these “things.” The second part of the chapter instructs you on how to build an entity-relationship diagram, and the final topic in the chapter is how to build a class diagram domain model.

Learning Objectives
After reading this chapter, you should be able to:
- Explain how the concept of “things” in the problem domain also defines requirements
- Identify and analyze data entities and domain classes needed in the system
- Read, interpret, and create an entity-relationship diagram
- Read, interpret, and create a domain model class diagram
- Understand the domain model class diagram for the RMO Consolidated Sales and Marketing System

Opening Case
Waiters On Call Meal-Delivery System (Part 2): Two points can be emphasized in this case. First the analyst is identifying those “informational things” that the system needs to keep track of. Those things define the problem domain classes and include restaurants, menu items, customers, orders, and so forth. This short case does not attempt to distinguish between classes and attributes. The other point is that the analyst continues to ask probing and detailed questions. He does this both to ensure he has identified all the classes, but also so that he understands all the details (attributes) that will eventually need to be defined.
“Things” in the Problem Domain

Key Terms

- **problem domain** – the specific area (or domain) of the user’s business need (or problem) that is within the scope of the new system
- **brainstorming technique** – a technique to identify problem domain objects in which developers work with users in an open group setting
- **noun technique** – a technique to identify problem domain objects by finding and classifying the nouns in a dialog or description
- **attributes** – descriptive pieces of information about things or objects
- **identifier or key** – an attribute the value of which uniquely identifies an individual thing or object
- **compound attribute** – an attribute that consists of multiple pieces of information but is best treated in the aggregate
- **association** – a term, in UML, that describes a naturally occurring relationship between specific things, sometimes called a relationship
- **relationship** – a term that describes a naturally occurring association between specific things, sometimes called an association
- **cardinality** – a measure of the number of links between one object and another object in a relationship
- **multiplicity** – a measure, in UML, of the number of links between one object and another object in an association
- **multiplicity constraints** – the actual numeric count of the constraints on objects allowed in an association
- **binary associations** – associations between exactly two distinct types of things
- **unary association** – an association between two instances of the same type of thing
- **ternary association** – an association between exactly three distinct types of things
- **n-ary association** – an association between n distinct types of things

A basic problem with data modeling is what to use for the generic term of items in the real world that a system analyst is trying to discover. The authors have chosen to simply call them “things.” Although “things” is a very ambiguous term that can be used to identify almost any real world item or abstract idea, that information is in fact what data modeling is trying to capture.

The term “problem domain” is an interesting word choice. Often when we think of a problem, we think that something is broken. However, in this context, the word problem simply means a need that requires a business solution. In other words, system developers are developing a solution for a business need or business problem. Hence the term problem domain, or the context and arena in which the
business problem exists.

**The Brainstorming Technique**

This technique is a joint effort between the analyst and the users. Figure 4-1 illustrates the many different types of things that should be considered. “Things” ranges from very tangible things such as books and vehicles to roles such as employees and customers to quite abstract things such as flights and orders. (You cannot observe a flight or an order with any of your five senses.)

Here are the steps to follow when using the brainstorming technique:

1. Identify a user and a set of use cases.
2. Brainstorm with the user to identify things involved when carrying out the use case—that is, things about which information should be captured by the system.
3. Use the types of things (categories) to systematically ask questions about potential things, such as the following: Are there any tangible things you store information about? Are there any locations involved? Are there roles played by people that you need to remember?
4. Continue to work with all types of users and stakeholders to expand the brainstorming list.
5. Merge the results, eliminate any duplicates, and compile an initial list.

Care should be taken to distinguish between classes and attributes. See the next sections.

**The Noun Technique**

The noun technique is a more mechanical approach to identifying classes, but it is also a powerful technique. The basic idea is to identify all the nouns, which are always some type of “thing,” and build the list of all these nouns. Then refine the list to remove duplicates and identify which are classes and which are attributes of classes.

Here are the steps to follow when using the noun technique:

1. Using the use cases, actors, and other information about the system—including inputs and outputs—identify all nouns.
2. Using other information from existing systems, current procedures, and current reports or forms, add items or categories of information needed.
3. As this list of nouns builds, you will need to refine it. Ask these questions about each noun to help you decide whether you should include it:
   - Ask questions, such as is it important and inside the scope, to see if it should be included.
   - Ask questions to see if it really should be excluded, such as is it only a report or an input or is it an attribute.
   - Ask questions to see if it needs further research. In other words that you cannot answer whether it needs to be included or excluded.
4. Create a master list of all nouns identified and then note whether each one should be included,
excluded, or researched further.

5. Review the list with users, stakeholders, and team members and then refine the list of things in the problem domain.

Attributes of Things

The noun technique involves listing all the nouns that come up in discussions or documents about the requirements. Many of these nouns are actually attributes. During the refinement of the nouns on the list of nouns, an evaluation of each noun will determine whether the noun is an independent item, e.g. a class, or whether it describes or characterizes another noun and hence is an attribute. One attribute may be used to identify a specific thing, such as a Social Security number for an employee or an order number for a purchase. The attribute that uniquely identifies the thing is called an identifier or key.

Associations Among Things

An association is a naturally occurring relationship between specific things, such as an order is placed by a customer and an employee works in a department. Is placed by and works in are two associations that naturally occur between specific things. Information systems need to store information about employees and about departments, but equally important is storing information about the specific associations.

In database management, the term relationship is often used in place of association, which is the term used when modeling in UML. The authors use association in this book because the emphasis is on UML diagrams and terms.

It is also important to understand the nature of each association in terms of the number of links for each thing. Cardinality can be one-to-one or one-to-many. The term multiplicity is used to refer to the number of links in UML and should be used when discussing UML models. Multiplicity is established for each direction of the association. It is important to describe not just the multiplicity but also the range of possible values of the multiplicity (the minimum and maximum multiplicity). These terms are referred to as multiplicity constraints. The next sections on the notation explain more about the various combinations of cardinality and multiplicity constraints.

Associations between two classes are called binary associations. Sometimes an association will be between different elements in the same class or set. These are called unary associations. Associations can also exist between multiple classes, such as three classes – a ternary association, or some arbitrary number n – n-ary associations.

The Entity-Relationship Diagram

Key Terms

- **data entities** – the term used in an ER diagram to describe sets of things or individual things
- **entity-relationship diagram (ERD)** – a diagram consisting of data entities (i.e., sets of things) and their relationships
- **semantic net** – a graphical representation of an individual data entity and its relationship with other individual data entities
A data model commonly used by traditional analysts and database analysts is called the entity-relationship diagram (ERD). The ERD is not a UML diagram, but it is very commonly used and is quite similar to the UML domain model class diagram.

Examples of ERD Notation

On the entity-relationship diagram, rectangles represent data entities, and the lines connecting the rectangles show the relationships among data entities. Cardinality constraints are expressed by the notations on the lines. Remember that the cardinality applies to each end of the relationship and contains and minimum and maximum value. Refer Figure 4-7 to note the bars, circles, and crows-feet that designate minimum and maximum cardinality values. A bar represents one, a circle represents zero, and crows-feet represent many.

Sometimes it is obvious what the relationships between data entities should be. Often it is also clear what the cardinality constraints should be. However, when it is not clear, creating a semantic net can be helpful. A semantic net is simply a drawing of the individual data entities showing what relationships can exist. For example in Figure 4-9, orders are always only connected to one customer, but a customer will have multiple orders. It is even possible to have a customer, such as Mary, that has not made any orders yet. A semantic net can be useful to help define cardinality constraints.

The Domain Model Class Diagram

Key Terms

- **class** – a category or classification of a set of objects or things
- **domain classes** – classes that describe objects from the problem domain
- **class diagram** – a diagram consisting of classes (i.e., sets of objects) and associations among the classes
- **domain model** – class diagram a class diagram that only includes classes from the problem domain
- **camelback notation or camelcase notation** – when words are concatenated to form a single word and the first letter of each embedded word is capitalized
- **association class** – an association that is also treated as a class; often required in order to capture attributes for the association
- **generalization/specialization relationship** – a type of hierarchical relationship in which subordinate classes are subsets of objects of the superior classes; an inheritance hierarchy
- **superclass** – the superior or more general class in a generalization/specialization relationship
- **subclass** – the subordinate or more specialized class in a generalization/specialization relationship
- **inheritance** – the concept that specialization classes inherit the attributes of the generalization class
- **abstract class** – a class that describes a category or set of objects but that never includes
individual objects or instances

- **concrete class** – a class that allows individual objects or instances to exist
- **whole-part relationship** – a relationship between classes in which one class is a part or a component portion of another class
- **aggregation** – a type of whole-part relationship in which the component parts also exist as individual objects apart from the aggregate
- **composition** – a type of whole-part relationship in which the component parts cannot exist as individual objects apart from the total composition

Current approaches to system development, primarily object-oriented approaches, use the term *class* rather than data entity and use concepts and notations based on UML to model the things in the problem domain. A *class* is a category or classification used to describe a set of objects. So a class is a category, but it is also a set of objects. Classes that describe things in the problem domain are called *domain classes*. Domain classes have attributes and associations. Multiplicity (called cardinality in an ERD) applies among classes.

On a *class diagram*, rectangles represent classes, and the lines connecting the rectangles show the associations among classes. The domain class symbol is a rectangle with two sections. The top section contains the name of the class, and the bottom section lists the attributes of the class. Class names are capitalized and attributes begin with lower case. Both use camelcase notation. Later, you will learn that the design class symbol includes a third section at the bottom for listing methods of the class; methods do not apply to problem domain classes.

**Domain Model Class Diagram Notation**

This section contains many examples of UML class diagram notation as explained above. **Multiplicity** is denoted by numbers placed next to the class at either end of a binary association line. Minimum and maximum multiplicity is denoted by 0..1 (zero or one) and 1..* (one/many), and so forth. Figure 4-13 summarizes multiplicity notation.

Many-to-many associations allow data objects from one class to be connected to many objects of the associated class, and this association occurs in both directions. Sometimes the association itself will need to have attributes associated with it. An example in the book (Figures 4-15 and 4-16) where a CourseSection will have many students and a Student will be registered in many course sections. The student also will receive a grade for that course section. The grade cannot be associated solely with the student, nor with the course section. The grade belongs to the association. In other words we need to treat the CourseSection—Student association as a class. As shown in Figure 4-16 that is done by a dashed line to a class called an *association class*. Don’t be surprised if you have some problems identifying association classes.

**More Complex Issues about Classes of Objects**

**Generalization/specialization relationships** are based on the idea that people classify things in terms of similarities and differences. A generalization/specialization relationship is used to structure these things from the more general to the more special. Each class of things in the hierarchy might have a more general class above it, called a *superclass*. At the same time, a class might have a more
specialized class below it, called a **subclass**. UML class diagram notation uses a triangle that points to the superclass to show a generalization/specialization hierarchy.

A generalization/specialization relationship is also a set/subset relationship. All of the objects in the specialized class are also within the generalized class. Thus all of the attributes in the generalized class are also attributes of the specialized class. This is called **inheritance**, since the subclass or subset inherits the attributes of the superclass or superset. Figure 4-18 is an example. Sometimes the superclass is defined only to allow inheritance of attributes, but has no actual real-life objects in it. In that situation, you call the superclass an **abstract class**, and is denoted by an italicized name. A class that does have objects is a **concrete class**.

**Whole-part relationships** are used to show an association between one class and other classes that are parts of that class. There are two types of whole-part relationships: **aggregation** and **composition**. Aggregation refers to a type of whole-part relationship between the aggregate (whole) and its components (parts), where the parts can exist separately and is represented by an open diamond. Composition refers to whole-part relationships that are even stronger, where the parts, once associated, can no longer exist separately, and is represented by a solid dark diamond. Whole-part relationships—aggregation and composition—mainly allow the analyst to express subtle distinctions about associations among classes. As with any association, multiplicity can apply.

**Assignment**

**End of Chapter Case**

**On the Spot Courier Services** (running case): On the Spot is a small, but growing, courier service that needs to track customers, package pickups, package deliveries, and delivery routes. The case description for this chapter is quite elaborate and describes both processing and data that will occur for the system. You are asked to use the noun technique and identify nouns that become classes and attributes, using both this narrative and the narrative in previous chapters. You then build relationships and a final class diagram with class, attributes, primary keys, relationships, and multiplicity constraints. You will create the class diagram using Visio.