Chapter 11

DOMAIN MODEL: ADDING ASSOCIATIONS

Objectives

- Identify associations for a domain model.
- Distinguish between need-to-know and comprehension-only associations.

Introduction

It is useful to identify those associations of conceptual classes that are needed to satisfy the information requirements of the current scenarios under development, and which aid in comprehending the domain model. This chapter explores the identification of suitable associations, and adds associations to the domain model for the NextGen case study.

11.1 Associations

An **association** is a relationship between types (or more specifically, instances of those types) that indicates some meaningful and interesting connection (see Figure 11.1).

In the UML associations are defined as "the semantic relationship between two or more classifiers that involve connections among their instances."

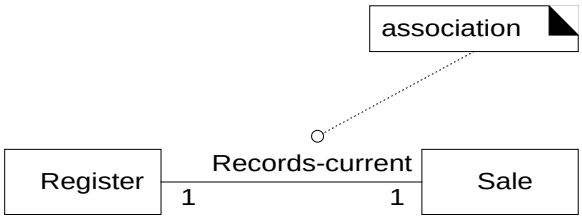

Figure 11.1 Associations.

Criteria for Useful Associations

Associations worth noting usually imply knowledge of a relationship that needs to be preserved for some duration—it could be milliseconds or years, depending on context. In other words, between what objects do we need to have some memory of a relationship? For example, do we need to remember what *SalenLineItem* instances are associated with a *Sale* instance? Definitely, otherwise it would not be possible to reconstruct a sale, print a receipt, or calculate a sale total.

Consider including the following associations in a domain model:
- Associations for which knowledge of the relationship needs to be preserved for some duration ("need-to-know" associations).
- Associations derived from the Common Associations List.

By contrast, do we need to have memory of a relationship between a current *Sale* and a *Manager?* No, the requirements do not suggest that any such relationship is needed. It is not wrong to show a relationship between *a Sale* and *Manager,* but it is not compelling or useful in the context of our requirements.

This is an important point. On a domain model with n different conceptual classes, there can be n-(n-l) associations to other conceptual classes—a potentially large number. Many lines on the diagram will add "visual noise" and make it less comprehensible. Therefore, be parsimonious about adding association lines. Use the criterion guidelines suggested in this chapter.

11.2 The UML Association Notation

An association is represented as a line between classes with an association name. The association is inherently bidirectional, meaning that from instances of either class, logical traversal to the other is possible.

This traversal is purely abstract; it is *not a* statement about connections between software entities.

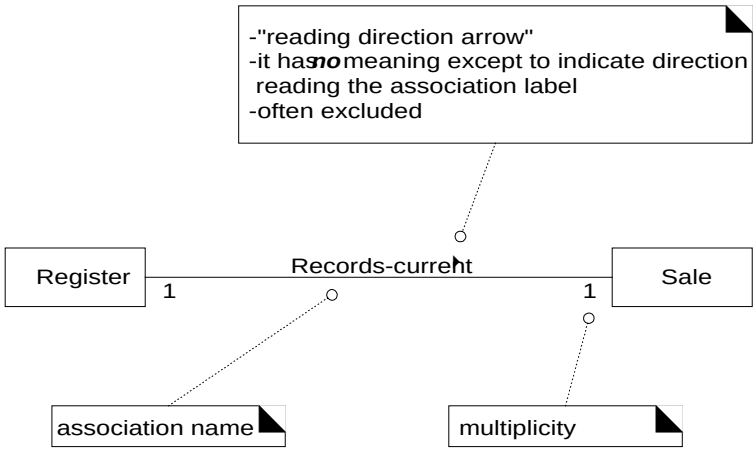

Figure 11.2 The UML notation for associations.

The ends of an association may contain a multiplicity expression indicating the numerical relationship between instances of the classes.

An optional "reading direction arrow" indicates the direction to read the association name; it does not indicate direction of visibility or navigation.

If not present, it is conventional to read the association from left to right or top to bottom, although the UML does not make this a rule (see Figure 11.2).

> The reading direction arrow has no meaning in terms of the model; it is only an aid to the reader of the diagram.

11.3 Finding Associations—Common Associations List

Start the addition of associations by using the list in Table 11.1.

It contains common categories that are usually worth considering. Examples are drawn from the store and airline reservation domains.

Category	Examples
A is a physical part of B	*Drawer — Register (or more specifically, a POST)* *Wing — Airplane*
A is a logical part of B	*SalesLineItem — Sale* *FlightLeg—FlightRoute*
A is physically contained in/on B	*Register — Store, Item — Shelf* *Passenger — Airplane*
A is logically contained in B	*ItemDescription — Catalog* *Flight— FlightSchedule*
A is a description for B	*ItemDescription — Item* *FlightDescription — Flight*
A is a line item of a transaction or report B	*SalesLineItem — Sale* *Maintenance Job — MaintenanceLog*
A is known/logged/recorded/reported/captured in B	*Sale — Register* *Reservation — FlightManifest*
A is a member of B	*Cashier — Store* *Pilot — Airline*
A is an organizational subunit of B	*Department — Store* *Maintenance — Airline*
A uses or manages B	*Cashier — Register* *Pilot — Airplane*
A communicates with B	*Customer — Cashier* *Reservation Agent — Passenger*
A is related to a transaction B	*Customer — Payment* *Passenger — Ticket*
A is a transaction related to another transaction B	*Payment — Sale* *Reservation — Cancellation*
A is next to B	*SalesLineItem — SalesLineItem* *City— City*

Category	Examples
A is owned by B	*Register — Store* *Plane — Airline*
A is an event related to B	*Sale — Customer, Sale — Store* *Departure — Flight*

Table 11.1 Common Associations List.

High-Priority Associations

Here are some high-priority association categories that are invariably useful to include in a domain model:

- A is a *physical or logical part* of B.
- A is *physically or logically contained* in/on B.
- A is *recorded in* B.

11.4 Association Guidelines

- Focus on those associations for which knowledge of the relationship needs to be preserved for some duration ("need-to-know" associations).
- It is more important to identify *conceptual classes* than to identify associations.
- Too many associations tend to confuse a domain model rather than illuminate it. Their discovery can be time-consuming, with marginal benefit.
- Avoid showing redundant or derivable associations.

11.5 Roles

Each end of an association is called a **role.** Roles may optionally have:

- name
- multiplicity expression
- navigability

Multiplicity is examined next, and the other two features are discussed in later chapters.

Multiplicity

Multiplicity defines how many instances of a class *A* can be associated with one instance of a class *B* (see Figure 11.3).

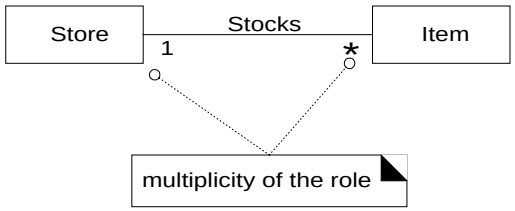

Figure 11.3 Multiplicity on an association.

For example, a single instance of a *Store* can be associated with "many" (zero or more, indicated by the *) *Item* instances.

Some examples of multiplicity expressions are shown in Figure 11.4.

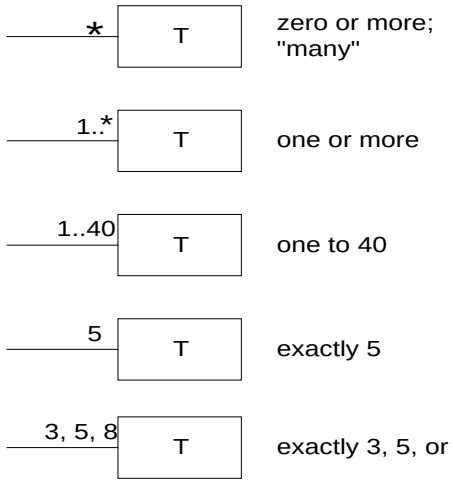

Figure 11.4 Multiplicity values.

The multiplicity value communicates how many instances can be validly associated with another, at a particular moment, rather than over a span of time. For example, it is possible that a used car could be repeatedly sold back to used car dealers over time. But at any particular moment, the car is only *Stocked-by* one dealer. The car is not *Stocked-by* many dealers at any particular moment. Similarly, in countries with monogamy laws, a person can be *Married-to* only one other person at any particular moment, even though over a span of time, they may be married to many persons.

The multiplicity value is dependent on our interest as a modeler and software developer, because it communicates a domain constraint that will be (or could be) reflected in software. See Figure 11.5 for an example and explanation.

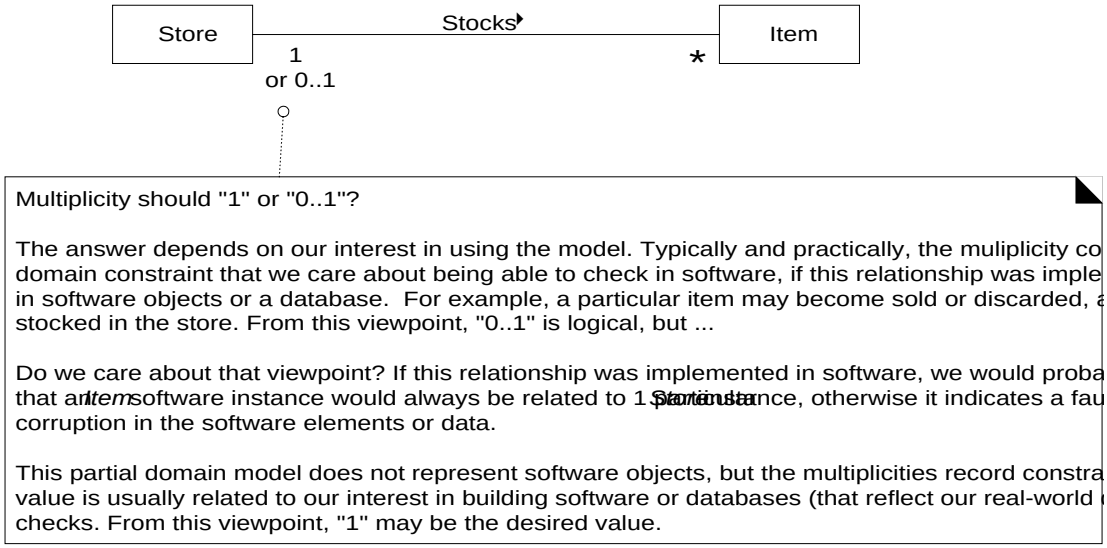

Figure 11.5 Multiplicity is context dependent.

Rumbaugh gives another example *of Person* and *Company* in the *Works-for* association [Rumbaugh91]. Indicating if a *Person* instance works for one or many *Company* instances is dependent on the context of the model; the tax department is interested in *many;* a union probably only *one.* The choice usually practically depends on whom we are building the software for, and thus the valid multiplicities in an implementation.

11.6 How Detailed Should Associations Be?

Associations are important, but a common pitfall in creating domain models is to spend too much time during investigation trying to discover them.

It is critical to appreciate the following:

11.7 Naming Associations

> Name an association based on a *TypeName-VerbPhrase-TypeName* format where the verb phrase creates a sequence that is readable and meaningful in the model context.

Association names should start with a capital letter, since an association represents a classifier of links between instances; in the UML, classifiers should start with a capital letter. Two common and equally legal formats for a compound association name are:

- *Paid-by*
- *PaidBy*

In Figure 11.6, the default direction to read an association name is left to right or top to bottom. This is not a UML default, but a common convention.

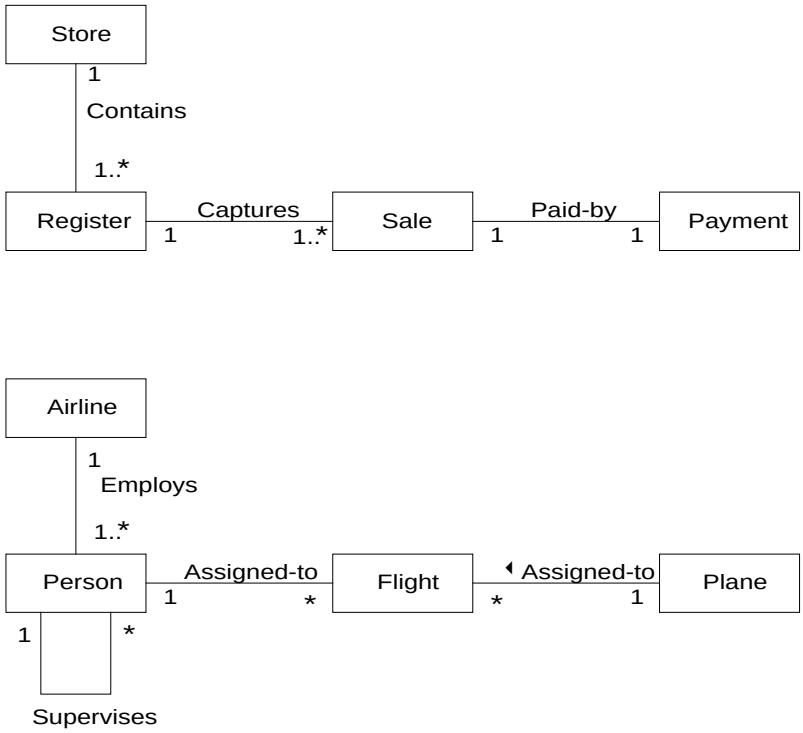

Figure 11.6 Association names.

11.8 Multiple Associations Between Two Types

Two types may have multiple associations between them; this is not uncommon. There is no outstanding example in our POS case study, but an example from the domain of the airline is the relationships between a *Flight* (or perhaps more precisely, a *FlightLeg)* and *an Airport* (see Figure 11.7); the flying-to and flying-from associations are distinctly different relationships, which should be shown separately.

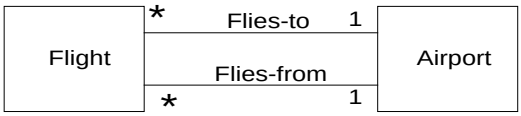

Figure 11.7 Multiple associations.

11.9 Associations and Implementation

During domain modeling, an association is *not* a statement about data flows, instance variables, or object connections in a software solution; it is a statement that a relationship is meaningful in a purely conceptual sense—in the real world. Practically speaking, many of these relationships will typically be implemented in software as paths of navigation and visibility (both in the Design Model and Data Model), but their presence in a conceptual (or essential) view of a domain model does not require their implementation.

When creating a domain model, we may define associations that are not necessary during implementation. Conversely, we may discover associations that need to be implemented but were missed during domain modeling. In these cases, the domain model can be updated to reflect these discoveries.

Suggestion
Should prior investigative models such as a domain model be updated with insights (such as new associations) revealed during implementation work? Do not bother unless there is some future practical use for the model. If it is just (as is sometimes the case) a temporary artifact used to provide inspiration for a later step, and will not be meaningfully used later on, why update it? Avoid making or updating any documentation or model unless there is a concrete justification for future use.

Later on we will discuss ways to implement associations in an object-oriented programming language (the most common is to use an attribute that references

an instance of the associated class), but for now, it is valuable to think of them as purely conceptual expressions, *not* statements about a database or software solution. As always, deferring design considerations frees us from extraneous information and decisions while doing pure "analysis" investigations and maximizes our design options later on.

11.10 NextGen POS Domain Model Associations

We can now add associations to our POS domain model. We should add those associations which the requirements (for example, use cases) suggest or imply a need to remember, or which otherwise are strongly suggested in our perception of the problem domain. When tackling a new problem, the common categories of associations presented earlier should be reviewed and considered, as they represent many of the relevant associations that typically need to be recorded.

Unforgettable Relationships in the Store

The following sample of associations is justified in terms of a need-to-know. It is based on the use cases currently under consideration.

Register Records Sale	To know the current sale, generate a total, print a receipt.
Sale Paid-by Payment	To know if the sale has been paid, relate the amount tendered to the sale total, and print a receipt.
ProductCatalog Records ProductSpecification	To retrieve an *ProductSpecification,* given an itemID.

Applying the Category of Associations Checklist

We will run through the checklist, based on previously identified types, considering the current use case requirements.

Category	System
A is a physical part of B	*Register — CashDrawer*
A is a logical part of B	*SalesLineItem — Sale*
A is physically contained in/on B	*Register — Store* *Item — Store*

Category	System
A is logically contained in B	*ProductSpecification — Product-Catalog* *ProductCatalog — Store*
A is a description for B	*ProductSpecification — Item*
A is a line item of a transaction or report B	*SalesLineItem — Sale*
A is logged/recorded/reported/captured in B	*(completed) Sales — Store* *(current) Sale — Register*
A is a member of B	*Cashier — Store*
A is an organizational subunit of B	*not applicable*
A uses or manages B	*Cashier — Register* *Manager — Register* *Manager — Cashier, but probably not applicable.*
A communicates with B	*Customer — Cashier*
A is related to a transaction B	*Customer — Payment* *Cashier — Payment*
A is a transaction related to another transaction B	*Payment — Sale*
A is next to B	*SalesLineItem — SalesLineItem*
A is owned by B	*Register — Store*

11.11 NextGen POS Domain Model

The domain model in Figure 11.8 shows a set of conceptual classes and associations that are candidates for our POS application. The associations were primarily derived from the candidate association checklist.

Preserve Only Need-to-Know Associations?

The set of associations shown in the domain model of Figure 11.8 were, for the most part, mechanically derived from the association checklist. However, it may be desirable to be more choosy in the associations included in our domain model. Viewed as a tool of communication, it is undesirable to overwhelm the domain

model with associations that are not strongly required and which do not illuminate our understanding. Too many uncompelling associations obscure rather than clarify.

As previously suggested, the following criteria for showing associations is recommended:

- Focus on those associations for which knowledge of the relationship needs to be preserved for some duration ("need-to-know" associations).
- Avoid showing redundant or derivable associations.

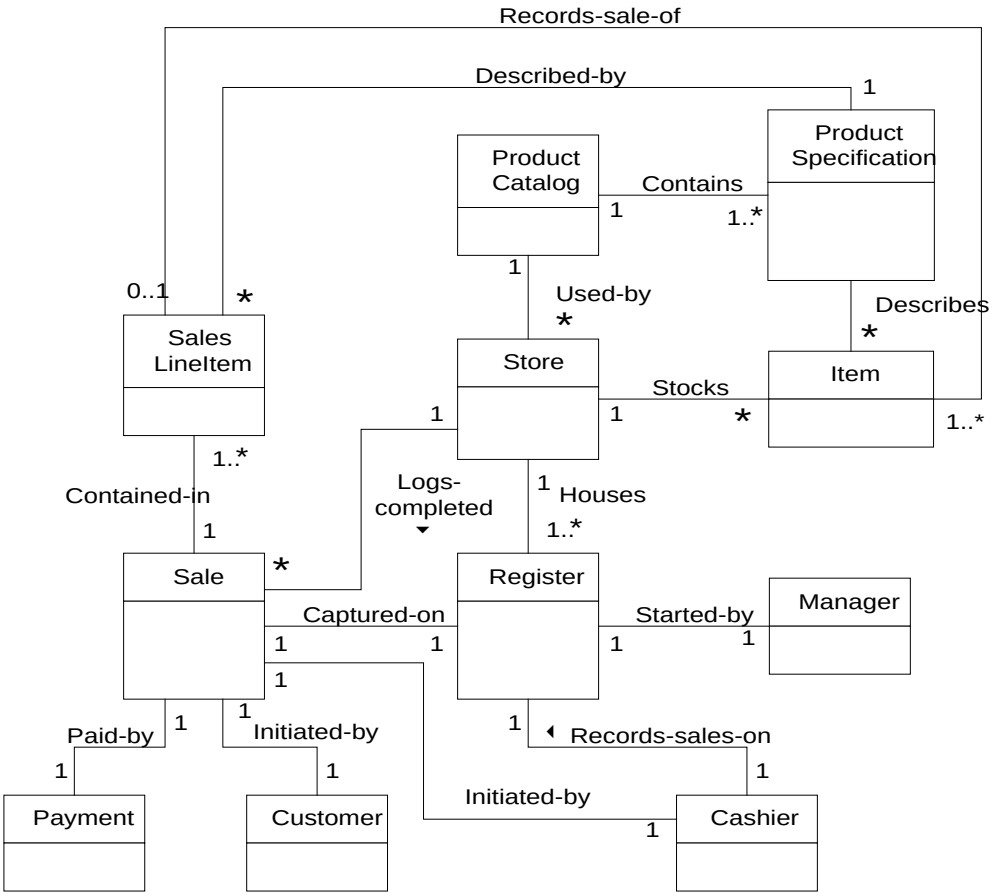

Figure 11.8 A partial domain model.

Based on this advice, not every association currently shown is compelling. Consider the following:

Association	Discussion
Sale Entered-by Cashier	The requirements do not indicate a need-to-know or record the current cashier. Also, it is derivable if the *Register Used-by Cashier* association is present.
Register Used-by Cashier	The requirements do not indicate a need-to-know or record the current cashier.
Register Started-by Manager	The requirements do not indicate a need-to-know or record the manager who starts up a *Register*.
Sale Initiated-by Customer	The requirements do not indicate a need-to-know or record the current customer who initiates a sale.
Store Stocks Item	The requirements do not indicate a need-to-know or maintain inventory information.
SalesLineItem Records-sale-of Item	The requirements do not indicate a need-to-know or maintain inventory information.

Note that the ability to justify an association in terms of need-to-know is dependent on the requirements; obviously a change in these—such as requiring that the cashier's ID show on a receipt—changes the need to remember a relationship.

Based on the above analysis, it *may* be justifiable to delete the associations in question.

Associations for Need-to-Know vs. Comprehension

A strict need-to-know criterion for maintaining associations will generate a minimal "information model" of what is needed to model the problem domain—bounded by the current requirements under consideration. However, this approach may create a model that does not convey (to us or anyone else) a full understanding of the domain.

In addition to being a need-to-know model of information about things, the domain model is a tool of communication in which we are trying to understand and communicate to others important concepts and their relationships. From this viewpoint, deleting some associations that are not strictly demanded on a

need-to-know basis can create a model that misses the point—it does not communicate key ideas and relationships.

For example, in the POS application: although on a strict need-to-know basis it might not be necessary to record *Sale Initlated-by Customer,* its absence leaves out an important aspect in understanding the domain—that a customer generates sales.

In terms of associations, a good model is constructed somewhere between a minimal need-to-know model and one that illustrates every conceivable relationship. The basic criterion for judging its value?—Does it satisfy all need-to-know requirements and additionally clearly communicate an essential understanding of the important concepts in the problem domain?

Emphasize need-to-know associations, but add choice comprehension-only associations to enrich critical understanding of the domain.

Chapter 12

DOMAIN MODEL: ADDING ATTRIBUTES

Any sufficiently advanced bug is indistinguishable from a feature.

—Rich Kulawiec

Objectives

- Identify attributes in a domain model.
- Distinguish between correct and incorrect attributes.

Introduction

It is useful to identify those attributes of conceptual classes that are needed to satisfy the information requirements of the current scenarios under development. This chapter explores the identification of suitable attributes, and adds attributes to the domain model for the NextGen domain model.

12.1 Attributes

An attribute is a logical data value of an object.

Include the following attributes in a domain model: Those for which the requirements (for example, use cases) suggest or imply a need to remember information.