Collaboration Rules

4

I am prepared to meet anyone, but whether anyone is prepared for the great ordeal of meeting me is another matter. ■

Mark Twain

Object-Oriented Business Rules

The previous chapter discussed the 12 collaboration patterns and explained their uses for modeling the people, places, things, and events in real-world domains. Such domains are frequently called business domains, although their realm may be non-business endeavors from government to entertainment. This book will use the terms "business domain" and "real-world domain" interchangeably, and will use the term "business rules" for constraints governing what actions and information are legal and valid within a business domain. Business rules are the heart of any domain. Without their constraining force, business processes may be initiated by the wrong people, at the wrong times and places, and with things improperly selected or illegally defined. Running business processes with illegal information not only corrupts system integrity, but also induces egregious real-world effects. Business rules prevent illegal product sales to under-age consumers and overseas customers; business rules prevent movement by an elevator with open doors; business rules define who can respond to a request-for-quota-

tion and enforce the deadline for quotes; and business rules determine which house of the U. S. Congress can initiate an appropriations bill or approve a treaty. Because business rules are so crucial to the success and goals of a business domain, it cannot be considered properly or completely modeled until its business rules are specified.

Capturing and specifying business rules is an analysis activity.[1] An ideal modeling methodology seamlessly integrates the business rules with the model, enabling them to be defined at the same time, and in the same vocabulary; integration simplifies and eases the process of keeping the rules and model in sync as one or the other evolves. Our modeling methodology achieves this integration by decomposing all business domains into pairs of collaborating objects, and transforming all business rules into tests between collaborating objects. These tests are called collaboration rules, and how to describe business rules as collaboration rules is the theme of this chapter.

Business Rules and Collaborations

Most business rules involve restrictions on which people can participate, what events can occur, where events can occur, what things can be involved, and how those things can be described, grouped, classified, and assembled. These relationships between people, places, things, and events are modeled by the 12 collaboration patterns; thus, a business domain modeled with collaboration patterns can easily express its business rules in terms of collaborating objects.

EXAMPLE—Business rules determine whether a product (`specific item`) can be added to an order (`transaction`), whether an elevator (`specific item`) can accept a request (`transaction`) to go to another floor, whether a bidder (`role`) can bid (`follow-up transaction`) on a request-for-quotation (`transaction`), and whether a bill (`specific item`) can be submitted (`transaction`) for the calendar (`place`) of a particular house of the U. S. Congress.

Collaboration Rules

Collaboration rules test whether two objects can create a new collaboration or dissolve their existing one; in more technical terms, collaboration rules establish pre-conditions for setting or removing an object connection.

1. Some organizations might consider this a design activity. In the extreme case, some might consider it a programming activity. This chapter lends proof to the argument it is an analysis activity.

Because these rules often run in response to user actions, a common mistake is to give the human user interface responsibility for checking them. This is a bad move.[2] If business rules are not in the business objects, every redesign and enhancement of the user interface puts them in jeopardy. Business rules lost in a user interface are not easily extracted and ported to other platforms and other technologies. Business rules kept in the business objects live on, through all manner of user interfaces, platforms, and new technologies. Updating business rules is also considerably easier when they live in the objects they govern; it is easy to find them, and to keep them in sync with changes in the objects they affect. Objects are responsible for their own welfare and should not be governed by outside forces.

Collaboration rules do not run when an object and its collaborators are being restored from a database, or a non-finalized collaboration is being rolled back, which frequently happens with event collaborations.[3] In the first case, objects being restored from persistent storage are presumed to have already passed through the business rules. In the second case, an event involves collaborations between multiple objects—people, places, and things; if any one collaboration should fail, then the event is not valid, and any previous collaborations established with it must be dissolved—checking business rules would, therefore, simply be gratuitous.

Transforming Business Rules into Collaboration Rules

Consistently transforming business rule constraints into collaboration rules requires: (1) deciding how to express the constraints, and (2) deciding how to distribute the rule checking between the two collaborating objects. Because collaboration rules involve two objects mutually deciding whether to make or dissolve a collaboration with the other, the first requirement boils down to deciding what information an object asks of itself and its potential collaborator. We believe that the following five categories handle all possible questions: type, property, state, collaborator multiplicity, and collaborator conflicts. These categories are discussed below.

The second requirement boils down to deciding which collaborator checks what rules, and herein lays the power of collaboration patterns. Because collaboration patterns represent real-world relationships, and each pattern player has a distinct meaning, knowing the pattern involved determines

2. Although putting these rules in the business objects requires making objects, trying them out, and then possibly throwing them away, that is a very small price to pay.
3. See p. 76, "Removing and Rolling Back Event Collaborations."

what rules are likely and how the work of checking them is to be distributed between the two pattern players. The remainder of this chapter describes the distribution of collaboration rules among the 12 collaboration patterns. The different types of collaboration rules are also summarized on page 336, "Five Kinds of Collaboration Rules."

Type Rules

Type rules limit the kinds of collaborators an object can have. Think of them as zoning laws: they keep away the wrong types of objects (for example, you can't put a business in a residential area). For simplicity, only one collaborator needs to enforce the type rule: "If it is the wrong type for me, then I am the wrong type for it." The simplest type rules can be depicted on the object model diagram by looking at the connected objects; however, some type rules are dynamic, relying on a derived type computed from the object's state, properties, or collaborations.

> EXAMPLE—A carton of perishable food can only be loaded into a refrigerated truck.

Type rules raise the possibility that a collaborator may need to be specialized into one or more subclasses. Indeed, if the collaborator has different types with clear differences in behaviors, properties, or collaborations, then the analyst can include multiple specializations in the object model.[4] Otherwise, the specialization of types is left to design because it affects efficiency, reuse, and optimization goals. Clearly written type rules assist and inform the designer.

Multiplicity Rules

Multiplicity rules regulate the number of collaborators of an object, and usually these rules are easily depicted on the object model diagram; however, an object can have more dynamic rules determined by its properties. Multiplicity rules come into play when an object tries to add or break a collaboration; an object may refuse to add or break a collaboration if doing so would put it above the maximum or below the minimum required for its existence.

> EXAMPLE—A system for delivering online interactive educational lectures limits the number of students that may enroll in each lecture. Different lectures may have different limits depending on the material and lecturer.

4. A specialization is a subtype that differs from its generalization in properties, behaviors, or collaborations. Often, specializations are implemented with subclasses, but they need not be.

Property Rules

Property rules come in two variations: validation and comparison. A validation rule verifies a property value against a standard that is not dependent on the properties of the other potential collaborator. The object with the clearest access to the standard is the owner of the collaboration rule. Alternatively, a comparison rule measures a property value of one object against one or more of the property values of the other potential collaborator. Comparison rules commonly occur in domains with age, residency, or security requirements. The object with the properties defining the acceptable values owns the collaboration rule. In both cases, property values evaluated can be either stored or calculated data.

> EXAMPLE—A customer cannot place an order if he lacks a valid name, billing address, or credit card. A perishable food product cannot collaborate with a shipping container whose average temperature is outside the product's minimum and maximum temperature range.

State Rules

State rules specify the proper states an object needs to be in to begin or end a collaboration. Basically, a state rule asks the question: Am I ready? State rules are often confused with property rules, but they differ in several ways. While validation property rules check for legal property values, state rules check for "proper" state; according to a state rule, an object in a legal state may still not be in the proper state to make or break the collaboration. Also, state rules are always internal and this distinguishes them from comparison property rules that examine property values in both objects.

> EXAMPLE—An order cannot be shipped if it has been cancelled. An auction cannot accept a new bid after the auction has expired.

Conflict Rules

Conflict rules define the conditions under which current collaborators protest gaining a new collaborator or losing an existing one. Conflict rules come in two varieties: collection and event rules. Collection conflict rules occur when an object with a group of similar collaborators polls the group for permission to add a new collaborator to the group or remove an existing one. Event conflict rules check for conflicts among the collaborators of an event; they run when assigning people, places, or things to the event. Conflict rules are themselves type, multiplicity, property, or state rules.

EXAMPLE—A flight arrival cannot be scheduled at a gate if it conflicts with the existing flight arrivals for the gate. A product cannot be added to an order if it conflicts with the order's customer, for example, the customer is under-age for the product.

Collaboration Rules Table

For each collaboration pattern we use a simple table (Table 4.1) to show the division of labor between the two pattern players for collaboration rule checking.

TABLE 4.1
Collaboration Rules Table

	Player 1	Player 2
Type		✓
Multiplicity	✓	✓
Property	✓	✓
State	✓	✓
Conflict		✓

Each column heading names a pattern player, and each row indicates a collaboration rule category. A checkmark indicates whether a pattern player can assume responsibility for that category of rules. Some rule categories can only be properly handled by one of the pattern players; other categories can be handled by both; and occasionally, neither pattern player handles a category of rules. The remainder of this chapter shows the collaboration rules table for each collaboration pattern. A unified table showing the collaboration rules for all 12 patterns is on page 337, "Pattern Player Collaboration Rules."

Rules for Entity Collaborations

Entity collaborations use two pattern players to model individual people, places, things, and aggregations of these. With entity collaborations, one of the pattern players is more particular—more specific (`role`, `specific item`), more local (`place`), or more detailed (`part`, `content`, `member`)—than the other. Good object think says put the behavior closest to the data and information, and for the entity patterns, that translates into putting the collaboration rules in the most specific, local, or detailed pattern player.

PRINCIPLE 21
MOST SPECIFIC, LOCAL, OR DETAILED OWNS THE RULE

For collaboration patterns involving people, places, and things, put the collaboration rules in the most specific, local, or detailed pattern player.

Rolling Back Entity Collaborations

If an entity plays in more than one collaboration pattern, then its existence may require multiple collaborations to be established simultaneously. When an entity requires multiple collaborators, the entity's failure to collaborate with one of them invalidates its collaborations to the others. The invalid collaborations are rolled back, and the entity ceases to exist. Rolling back a collaboration is not the same as dissolving it, which requires checking business rules. With a rollback, the failure of the entity to complete all the collaborations necessary for its existence prompted the roll back, and destroying the failed entity should not require business rule checking, as it does not alter persistent business data.

EXAMPLE—A proposal generation system constructs proposals by selecting chapters from a proposal template. A chapter reference is created from the chapter and added as part of the proposal. The chapter reference acts as both a specific item to the chapter and a part of the proposal, and requires both objects to exist. The chapter reference's collaboration to its chapter is established first, followed by its proposal collaboration. If the proposal collaboration fails, then the chapter collaboration is rolled back, and the chapter reference ceases to exist.[5]

Actor – Role

Actor – Role Collaboration Rules

These collaboration rules test whether a `role` can be added or removed from an `actor`. Conceptually, the `role` is representing the `actor` in a specialized context, and as its representative, the `role` presents the `actor's` properties, behaviors, and collaborations as its own. This familiarity with the details of the `actor` and the special requirements of each context are reasons why the `role` does all the rule checking in this collaboration, as shown in Table 4.2. In a nutshell, each context has different rules, so each context `role` checks the rules.

5. See page 246, "`Item Assemblies`," and Figure 9.15.

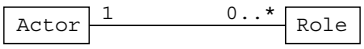

FIGURE 4.1
The `actor - role` pattern.

TABLE 4.2
`Actor - Role` Collaboration Rules

	Actor	Role
Type		✓
Multiplicity		✓
Property		✓
State		✓
Conflict		✓

Type Rules

RULE 1 A `role` knows the types of `actors` it can represent.

A `role` depends on its `actor`, so it has the burden of deciding if the `actor` is appropriate for it. Essentially, the `role` is checking that the `actor` has the correct interface to support its interactions in its context.

Multiplicity Rules

RULE 2 A `role` knows how many of its kind an `actor` can know.

RULE 3 A `role` knows exactly one `actor`, and it can never change its `actor`.

Occasionally, a domain allows an `actor` to support multiple `roles` for the same context, and hence the same type. As this rule is contextual, it belongs in the `role`. The personal and individual nature of a `role` prevents its transfer to another `actor`.

Property Rules

RULE 4 A `role` checks that the `actor` has valid property values.

Only the `actor's` properties necessary for the `role's` interactions are checked. There are several reasons why the `role` does the checking instead of the `actor`. Firstly, the `role` depends on these property values to carry out

its interactions in the context. Secondly, what is "proper" and "legal" may depend on the context. Finally, the `role` may be specialized, and each specialization may require checking additional properties; having each `role` check the properties it needs is preferable to having the `actor` know a different set of rules for each variety of the `role`.

State Rules

RULE 5 A `role` checks that the `actor` is in the proper state.

RULE 6 A `role` checks its own state before breaking with its `actor`.

As with the property rules, the `role` does the checking since the proper state for an `actor` depends on the context. As it interacts in its context, the `role` gains its own state reflective of its ongoing business, so before breaking with its `actor` and becoming invalid, a `role` checks its state to ensure it has no outstanding business.

Conflict Rules

RULE 7 A `role` enforces rules that define conflicts with other `roles`.

Within some domains, certain `roles` are either/or with respect to the same `actor`. In other words, the `actor` can take on one or the other `role`, but not both. These scenarios happen when there are interdependencies among the contexts. Contexts know if they conflict, so this knowledge is more appropriate for the `role` than the `actor`.

Actor – Role Collaboration Rules Example

EXAMPLE—An online office supply store takes orders from home, business, and government customers, and allows individual service suppliers to advertise and take orders for services such as resume writing, document preparation, and logo design. See Figure 4.2.

Collaboration Rules

Type
R1. [Role] A government customer or business customer requires an organization as its `actor`. A home customer or service supplier requires a person as its `actor`.

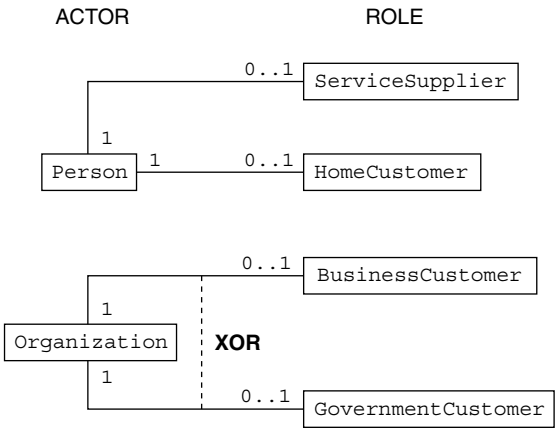

FIGURE 4.2
The `actors` and `roles` for an online office supply store.[6]

Multiplicity

R2. [Role] A home customer cannot collaborate with a person that already has a home customer `role`. A service supplier cannot collaborate with a person that already has a service supplier `role`. A government customer cannot collaborate with an organization that already has a government customer `role`. A business customer cannot collaborate with an organization that already has a business customer `role`.

R3. [Role] A home customer or service supplier cannot replace its person collaborator with another person. A government customer or business customer `role` cannot replace its organization collaborator with another organization.

Property

R4. [Role] A government customer requires its organization to have a valid name, government ID number, and contact telephone number. A business customer requires its organization to have a valid name and contact telephone number. A home customer requires its person to be 18 or older, and to have a valid name and email address. A service supplier requires its person to be 21 or older, and to have a valid name, email address, and telephone number.

6. The "XOR" marking in Figure 4.2 indicates that an organization can collaborate with either the business customer or the government customer, but not both.

State

R5. [Role] A business customer or government customer cannot collaborate with a closed or inactive organization.

R6. [Role] A business, home, or government customer cannot be removed if it has unpaid or undelivered orders. A service supplier cannot be removed if it has pending service orders.

Conflict

R7. [Role] A business customer cannot collaborate with an organization that is already a government customer, and vice versa.

Outer Place – Place

Outer Place – Place Collaboration Rules

With the `actor - role` pattern, all collaboration rules depend on a context of interaction, and consequently, are handled by the `role`. With the `outer place - place` pattern, both pattern players are in the same context, but one is more local than the other. As local objects tend to be more detailed and in greater varieties than global objects, the bulk of the work is pushed onto the local ones; accordingly, the `place` handles more of the collaboration rules in this pattern, as shown in Table 4.3.

TABLE 4.3
`Outer Place - Place` Collaboration Rules

	Outer Place	Place
Type		✓
Multiplicity	✓	✓
Property	✓	✓
State	✓	✓
Conflict		✓

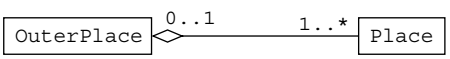

FIGURE 4.3
The `outer place - place` pattern.

Type Rules

RULE 8 A `place` knows the types of `outer places` that can house it.

Multiplicity Rules

RULE 9 An `outer place` may have an upper limit to the number of `places` it contains.

RULE 10 A `place` knows at most one `outer place`.

Instead of keeping a constant upper limit to the number of `places`, many `outer places` use property rules to keep within their limits. Not all `places` are in permanent locations. Some can move to new locations, and while in transition, exist without an `outer place`.[7]

Property Rules

RULE 11 A `place` uses property rules to avoid being located in an improper `outer place`.

RULE 12 An `outer place` uses property rules to constrain the `places` that go inside it.

Recall that with property rules, the object holding the standard or acceptable ranges owns the rule. With `outer place` and `place`, either one could have environmental constraints. An `outer place` may limit the maximum size of a `place` while a `place` may require its `outer place` to maintain an acceptable temperature.

State Rules

RULE 13 An `outer place` checks its own internal state before it allows a new `place` to be added or an old one to be removed.

RULE 14 A `place` checks its own internal state before it enters or leaves its `outer place`.[8]

Conflict Rules

RULE 15 A `place` enforces rules defining conflicts with other `places`.

Prior to adding a new `place`, an `outer place` asks its current `place` collaborators if any one of them conflicts with the proposed new `place`.

7. If y'all think like us, you may have imagined a mobile home almost immediately.
8. If your mobile home is still hooked to the main gas line, make sure it is unhooked before you pull away.

Outer Place – Place Collaboration Rules Example

EXAMPLE—Loading areas are locations within a warehouse where deliveries are deposited.[9] A loading area is either room temperature, refrigerated, or freezing, and contains smaller areas called loading bins that hold the goods and materials delivered to the loading area. A loading bin is capable of holding food, toxic materials, or non-toxic materials. Periodically, loading areas are reconfigured by removing old and adding new loading bins. Business rules define the legal partitions of a loading area into loading bins. See Figure 4.4.

Collaboration Rules

Type

R8. [Place] A loading bin knows the types of loading areas—room temperature, refrigerated, or freezing—that can house it.

Multiplicity

R9. [Outer Place] A loading area contains at least one loading bin.

R10. [Place] A loading bin is always within one loading area, being moved between loading areas, or not in use.

Property

R11. [Outer Place] A loading area cannot add a loading bin whose size is greater than its available space.

R12. [Place] A loading bin cannot exist in a loading area whose average temperature is not within its acceptable temperature range.

State

R13. [Outer Place] While it is receiving a delivery, a loading area cannot add or remove loading bins.

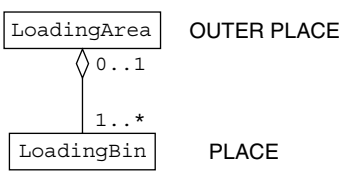

FIGURE 4.4
A LoadingArea is an `outer place` containing LoadingBins (`places`).

9. Loading areas and loading bins are `places` because they are locations where events occur (see p. 32, "Outer Place – Place," and p. 43, "Transaction – Place").

R14. [Place] While it contains goods, a loading bin cannot be removed from its loading area.

Conflict
R15. [Place] A loading bin designated for food conflicts with a loading bin designated for toxic materials. One of these cannot be added to a loading area if one or more of the other is already part of the loading area.

Item – Specific Item

Item – Specific Item Collaboration Rules

The `item - specific item` pattern seems a lot like the `actor - role` pattern. Even though the two pattern players are in the same context, one represents a generic object, while the other represents something more specific. As usual, most of the rule checking falls to the more specific pattern player, as shown in Table 4.4.

TABLE 4.4
`Item - Specific Item` Collaboration Rules

	Item	Specific Item
Type		✓
Multiplicity	✓	✓
Property		✓
State	✓	✓
Conflict		✓

Type Rules

RULE 16 A `specific item` collaborates with only one type of `item`.

Multiplicity Rules

RULE 17 An `item` knows about zero or more `specific items`.

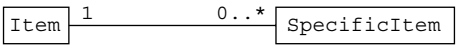

FIGURE 4.5
The `item - specific item` pattern.

RULE 18 A `specific item` knows exactly one `item`.

An `item` may have an upper limit to the number of its `specific items`. Often, an `item` acts as an object factory for its `specific items`, and the multiplicity check is executed prior to creating a new `specific item`. As with a `role`, a `specific item` is invalid without its `item`, and is not transferable to another.

Property Rules

RULE 19 A `specific item` must validate its properties before it can collaborate with an `item`.

A `specific item` does not check the properties of an `item` as a `role` checks the properties of an `actor` because the validities of its properties are not contextually dependent on the `specific item`, as an `actor's` are dependent on its `role`. On the other hand, while an `item` may act as the object factory for `specific items` and create each one from a given data set, the rules for validating the data set values belong in the `specific item`.[10]

State Rules

RULE 20 An `item` checks its state before adding a `specific item`.

RULE 21 A `specific item` checks its state prior to removing itself from its `item`.

A discontinued or obsolete `item` should not add new `specific items`, and a `specific item` involved in ongoing business should not let itself be removed from its `item`.

Conflict Rules

RULE 22 A `specific item` enforces rules defining conflicts with other `specific items`.

Typically, these are rules defining uniqueness conditions, ensuring that there are not two identical `specific items` for the same `item`.

10. Notice that both a `role` and a `specific item` must validate their properties before collaborating with the object on which they are dependent. This follows because the `role` and `specific item` do not exist prior to these collaborations, and so the validity of their properties has not yet been established. In the upcoming aggregation patterns the parts exist independently of the aggregate, requiring them to have valid properties even before they join it. Also, as the aggregate and part are in the same context, it does not make sense for one to validate the properties of the other.

Item – Specific Item Collaboration Rules Example

EXAMPLE—A video store puts tracking numbers on each videotape copy of a video title. Each video title has a unique identifier that is the root of the tracking numbers for its videotapes; thus, if the tracking number is unique to the video title, then it is unique to the store. A videotape has the following lifecycle states: uncirculated, in circulation, and out of circulation. A video title may be pulled from the shelf, taking all its videotapes out of circulation. See Figure 4.6.

Collaboration Rules

Type

R16. [Specific Item] A videotape requires a video title to supply its description.

Multiplicity

R17. [Item] A video title knows zero or more videotapes.

R18. [Specific Item] A videotape knows exactly one video title, and cannot replace it with another. The video title describes the videotape's title, production date, cast, and rating.

Property

R19. [Specific Item] A videotape must have a valid tracking number before it can collaborate with a video title, and hence belong to the video store.

State

R20. [Item] A discontinued video title cannot add a collaboration with a new videotape.

R21. [Specific Item] A videotape cannot be removed from its video title unless the title's state is "uncirculated."

Conflict

R22. [Specific Item] A videotape protests if its video title tries to add another videotape with the same tracking number.

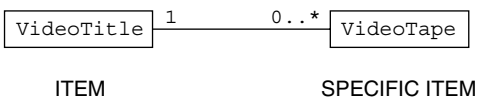

FIGURE 4.6
A VideoTitle knows about some number of VideoTape copies.

Assembly – Part

Assembly – Part Collaboration Rules

The assembly - part collaboration is one of three patterns for modeling aggregations of things. All three of these patterns follow the same principle for distributing collaboration rules between the pattern players: Put the bulk of the rules in the more specific pattern player, which is the part. This same principle applies to outer place - place, where the more global outer place is treated as more generic than the local place. In assembly - part, the assembly is considered more generic because it derives many of its properties by rolling up the properties of its parts, thus smoothing and averaging their particular details. Accordingly, the more specific part carries more of the load in checking collaboration rules, as shown in Table 4.5.

TABLE 4.5
Assembly - Part Collaboration Rules

	Assembly	Part
Type		✓
Multiplicity	✓	✓
Property	✓	✓
State	✓	✓
Conflict		✓

Type Rules

RULE 23 A part knows which types of assemblies can contain it.

Multiplicity Rules

RULE 24 An assembly must know at least one part.

RULE 25 A part knows about at most one assembly.

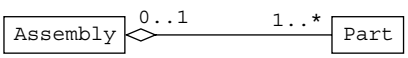

FIGURE 4.7
The assembly - part pattern.

Typically, things that contribute to an `assembly` can exist independently of it; however, once placed into an `assembly`, a `part` cannot be in another `assembly`.

Property Rules

RULE 26 An `assembly` uses property rules to keep out `parts` that do not meet its requirements.

RULE 27 A `part` uses property rules to avoid being placed inside an `assembly` that does not meet its requirements.

An `assembly` may require that a `part` conform to certain physical, logical, or business constraints, and a `part` may have its own physical, logical, or business constraints for selecting or rejecting an `assembly`.

State Rules

RULE 28 An `assembly` checks its state, allowing the addition or removal of a `part`.

RULE 29 A `part` checks its state before agreeing to join or leave an `assembly`.

Often an `assembly` has different lifecycle states; some states permit adding and removing `parts`, while other states lock the `assembly`, preventing the addition and removal of `parts`.

Conflict Rules

RULE 30 A `part` enforces rules defining conflicts between it and any other `parts`.

Before a `part` is added to an `assembly`, it must verify that it is compatible with the other `parts` in the `assembly`.

Assembly – Part Collaboration Rules Example

EXAMPLE—A made-to-order computer system is assembled from off-the-shelf components. Systems are either workstations or servers. As a component is added, a factory-floor tracking system records the serial numbers of the component added, reports any incompatibilities between the component and the system, and suggests a more compatible component. After a system is successfully completed according to its specification and approved by an inspector, the assembled components are decremented from inventory. See Figure 4.8.

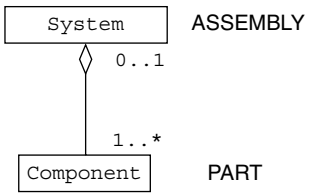

FIGURE 4.8
A made-to-order System and its core Components.

Collaboration Rules

Type

R23. [Part] A component knows what type of system, a server or a workstation, it can join. Some components work in both types of systems.

Multiplicity

R24. [Assembly] A system has at least one component.

R25. [Part] A component can be part of at most one system; however, if removed from a system, it can be added to another system later, or it can be sold individually.

Property

R26. [Assembly] A system cannot add a component whose price or weight would put the system over its maximums for those characteristics.

R27. [Part] A component with domestic electrical requirements cannot go into a system intended for overseas use.

State

R28. [Assembly] A system approved by an inspector cannot add or remove a component unless that approval is rescinded.

R29. [Part] A component in a damaged or defective state cannot be added to a system.

Conflict

R30. [Part] A component checks that it does not have incompatibilities with the other components in the system.

Container – Content

Container – Content Collaboration Rules

The container - content collaboration is another pattern for modeling aggregations of things. Many detailed and varied things can be placed in a plain brown box, and so like a box, a container is considered more generic than its contents. As with the other aggregate patterns, the more specific pattern player does more of the work of checking business rules, as shown in Table 4.6.

TABLE 4.6
Container - Content Collaboration Rules

	Container	Content
Type		✓
Multiplicity	✓	✓
Property	✓	✓
State	✓	✓
Conflict		✓

Type Rules

RULE 31 A content knows which types of containers can contain it.

Multiplicity Rules

RULE 32 A container may have an upper limit to the number of contents.

RULE 33 A content knows at most one container.

Instead of keeping a constant upper limit to the number of contents, many containers use property rules to keep within their limits. Contents, like parts, can exist outside of their containers.

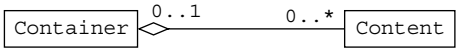

FIGURE 4.9
The container - content pattern.

Property Rules

RULE 34 A `container` uses property rules to keep out `contents` that do not meet its requirements.

RULE 35 A `content` uses property rules to avoid being placed inside a `container` that does not meet its requirements.

These rules include physical, logical, and business requirements.

State Rules

RULE 36 A `container` checks its state before allowing the addition or removal of a `content`.

RULE 37 A `content` checks its state before agreeing to enter or leave a `container`.

Conflict Rules

RULE 38 A `content` enforces rules defining conflicts between it and any other `contents`.

Before a `content` enters a `container`, it verifies that it does not conflict with other `contents` already in the `container`.

Container – Content Collaboration Rules Example

EXAMPLE—In a distribution center for a manufacturing plant, cases of product are removed from inventory, placed onto pallets, and loaded onto trucks for delivery to one or more distributors. Pallets come in two types, those designed for refrigerated trucks, and those not. See Figure 4.10.

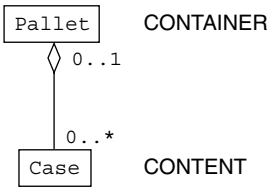

FIGURE 4.10
Cases are contained within Pallets.

Collaboration Rules

Type
R31. [Content] A case knows which type of pallet, refrigerated or non-refrigerated, can hold it.

Multiplicity
R32. [Container] The local trucking union, whose truck drivers are paid according to the number of pallets delivered, has negotiated that a pallet cannot contain more than 25 cases.

R33. [Content] A case can be placed on at most one pallet.

Property
R34. [Container] A pallet refuses any case whose weight puts it over its maximum limit.

R35. [Content] A case that is part of a rush order refuses to be placed on a pallet that is not scheduled to be loaded onto a truck within the next 24 hours.

State
R36. [Container] A pallet refuses to add or remove cases once it is loaded on a truck.

R37. [Content] An empty, damaged, defective, or expired case cannot be stored on a pallet.

Conflict
R38. [Content] A case of food cannot be stored on a pallet with a case of non-food product.

Group – Member

Group – Member Collaboration Rules

The `group - member` collaboration is the last on our list of patterns for modeling aggregations of things. Unlike the other two, a `group` does not have exclusive dominion over its constituent members; `members` can belong to multiple `groups`, which puts the burden on the `group` to check for conflicts with other `groups`, as shown in Table 4.7.

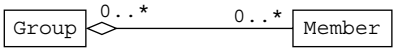

FIGURE 4.11
The `group - member` pattern.

TABLE 4.7
`Group - Member` Collaboration Rules

	Group	Member
Type		✓
Multiplicity	✓	✓
Property	✓	✓
State	✓	✓
Conflict	✓	✓

Type Rules

RULE 39 A `member` knows which types of `group` can contain it.

Multiplicity Rules

RULE 40 A `group` may have an upper limit to the number of `members` it can contain.

RULE 41 A `member` may have an upper limit to the number of `groups` it can join.

Property Rules

RULE 42 A `group` can require `members` to have properties within certain ranges or conform to certain standards.

RULE 43 A `member` can require `groups` to have properties within certain ranges or conform to certain standards.

State Rules

RULE 44 A `group` checks its own state before allowing a new `member` to be added or removed.

RULE 45 A `member` checks its own state before allowing itself to be added to or removed from a `group`.

Conflict Rules

RULE 46 A `group` enforces rules defining conflicts between it and any other `groups`.

RULE 47 A `member` enforces rules that define conflicts between it and any other `members`.

Group – Member Collaboration Rules Example

EXAMPLE—A luxury goods shopping site has a product catalog with shopping, promotional, and bargain categories. Shopping categories are ad hoc categories created to help the customer in browsing the site, and have names such as "sportswear," "pets," and "home and garden." Promotional categories are created to group and highlight unrelated products for seasonal events such as spring, school openings, and holidays. Bargain categories list products whose prices have been slashed appreciably. To protect the uniqueness of their products and brands, manufacturers can restrict the categories in which their products appear. See Figure 4.12.

Collaboration Rules

Type

R39. [Member] A product knows the types of catalog categories—shopping, promotional, or bargain—that can contain it. Some products cannot be placed into promotional or bargain groups due to manufacturer prohibitions intended to protect their branding.

Multiplicity

R40. [Group] A catalog category can limit the number of products it contains.

R41. [Member] A product can limit the number of catalog categories in which it appears.

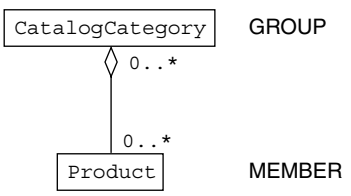

FIGURE 4.12
A Product can belong in multiple CatalogCategories.

Property

R42. [Group] A catalog category can require that all its products exhibit one or more colors, or have prices within a given range.

R43. [Member] To project uniqueness, a product can refuse to belong to any catalog category that is not limited in size, or that has an upper limit greater than is acceptable to the product.

State

R44. [Group]. A discontinued or expired catalog category cannot add or remove a product.

R45. [Member] A discontinued product cannot be added to a catalog category.

Conflict

R46. [Group] For branding purposes, some catalog categories are mutually exclusive so that product in one should not appear in the other, for example, a "his" category and a "hers" category. Before adding a product, a catalog category checks that it does not conflict with the product's existing categories.

R47. [Member] A product can have manufacturer's restrictions that prohibit it from appearing in a catalog category with products from a competitor.[11]

Rules for Event Collaborations

Distributing Event Collaboration Rules

In entity collaborations, which use two objects to model a real-world entity, the collaborators are of the same kind—two people, two places, or two things. Having two collaborators of the same kind allows one to be designated as more particular than the other and responsible for more of the work. Finding the most particular does not work with event collaborations where one collaborator represents a real-world entity, but the other represents a real-world interaction. Instead, the choice boils down to putting the rules in the interaction or the entity undergoing the interaction. Since an entity participates in many interactions, the temptation is to build dumb entities with smart interactions knowing the rules. This is tempting, but it is bad object think as the entity is the one doing the interacting, has the proper-

11. When this happens, manufacturers must pay to have an exclusive category.

ties that shape the interaction, and is the one permanently affected by the interaction. The principle with event collaborations is to allow the entity involved to decide whether or not to interact.

PRINCIPLE 22
INTERACTING ENTITY OWNS THE RULE

For collaboration patterns involving people, places, and things interacting with an event, each interacting pattern player that represents an entity owns its collaboration rules.

Removing and Rolling Back Event Collaborations

An entity's event collaborations reveal its history and are often a rich source for data mining. Therefore, in most business domains, removing a `transaction` from an entity is not permitted. Canceling an entity's real-world interaction is handled by changing the `transaction's` state to cancelled or voided and not by removing its object-world `transaction`.

An interaction is not complete until all the entities involved agree to participate in the interaction. If one collaborator refuses to accept the interaction, then the event must roll back the collaborations it made with other entities. Recall that rolling back a collaboration is not the same as dissolving it, and does not require business rule checking.[12] The failure of the interaction to complete prompted the roll back, and destroying the failed interaction should not require business rule checking, as it does not alter persistent business data.

EXAMPLE—A vehicle inspection (`transaction`) represents an interaction between a vehicle (`specific item`), inspection facility (`place`), and vehicle driver (`role`). If the collaboration between the vehicle inspection and vehicle driver should fail, then the collaborations between the inspection, the vehicle, and the facility are rolled back, and the vehicle inspection object is destroyed. The same behavior occurs if one of the other collaborations fails; the other two are rolled back. See Figure 4.13.

12. See p. 57, "Rolling Back Entity Collaborations."

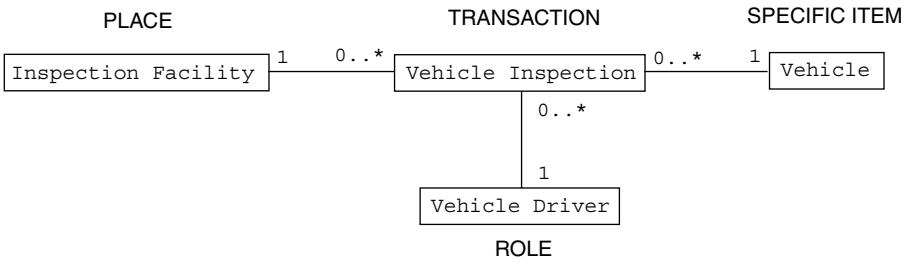

FIGURE 4.13
A VehicleInspection is not finalized if any of its collaborators reject it.

Transaction – Role

Transaction – Role Collaboration Rules

The `transaction - role` collaboration is the first of the event patterns, and its `role` represents the entity—the person, place, or thing—initiating an interaction in the real world represented by the `transaction`. Putting collaboration rules in the `transaction` looks tempting, as more `transactions` can easily be added for the same `role`; however, attaching many different kinds of `transactions` to a single type of `role` defeats the meaning of context.[13] Unrelated `transactions` that are not in the same context should not have the same type of `role`. Moreover, putting the rules in the `transaction` reduces a `role` to a mere data structure. Using a `role` to contain contextual data is good for data normalization, but without behavior, it is very bad object think. There are other ways of getting reuse while still keeping the rules with the `role`. Conclusion: The `role` does most of the work of checking its requirements for a `transaction`, as shown in Table 4.8.

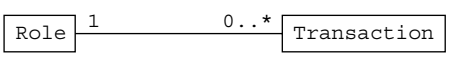

FIGURE 4.14
The `transaction - role` pattern.

13. "The set of circumstances or events in which a particular event occurs; situation." *The WordSmyth Educational Dictionary / Thesaurus (WEDT), http://www.wordsmyth.net*.

TABLE 4.8
`Transaction` - `Role` Collaboration Rules

	Transaction	Role
Type		✓
Multiplicity	✓	✓
Property		✓
State		✓
Conflict		✓

Type Rules

RULE 48 A `role` knows the kind of `transactions` in which it can participate.

Again, the `role` was created for the sole purpose of participating in these `transactions`; it darn well should know their types.

Multiplicity Rules

RULE 49 A `transaction` requires exactly one `role`; once set, the `role` cannot be removed without destroying the `transaction`.

RULE 50 A `role` knows zero to many `transactions`; typically, a `role` cannot remove a `transaction`.

The `role` is essential to the `transaction` because it is the real-world doer of the interaction represented by the `transaction`.[14] Allowing a `role` zero to many `transactions` gives it the option of interacting many times, or not at all. A `role`'s `transactions` reveal its history; therefore, `transactions` are usually not removed.

Property Rules

RULE 51 A `role` can require that `transactions` have properties within certain ranges or conform to certain standards.

The `role` decides if the `transaction` is too big, too late, or in any other way beyond the limits within which the `role` can interact.

14. See p. 41, "Transaction – Role."

State Rules

RULE 52 A `role` checks its state before accepting a `transaction`.

Inactive, expired, or dead `roles` should not take on new `transactions`.

Conflict Rules

RULE 53 A `role` checks for conflicts between its existing `transactions` and a new `transaction`. A `role` also checks for conflicts between itself and the other collaborations of a `transaction`.

A `role` can refuse a new `transaction` if it overlaps with the `role's` existing `transactions`; this happens when the `role` knows it cannot double-book itself by getting involved in coinciding events. Since the decision to permit or not permit double-booking lies in the `role`, it owns the conflict rule. A `role` can also refuse a `transaction` if its `specific item` is not suitable for the `role`; this occurs when the `role` has filters, for example, no green M&Ms, for screening out undesirable `specific items`. Since the filter is in the `role`, it owns the conflict rule. On the other hand, if the `specific item` has the standard, for instance, no one under 21, then it owns the rule.[15] The same principle applies for deciding conflict rules between the `role` and the `transaction's place`.[16]

Transaction – Role Collaboration Rules Example

EXAMPLE—A public library allows patrons to place books and other resources on hold at its various library branches. Resources are either circulating or restricted, and can have retrieval or usage fees. A restricted resource can only be held for a researcher patron. A regular patron is limited to five holds per month, and a researcher patron is allowed an unlimited number of holds. An open-ended resource hold is active until the patron collects the resource at which time it is completed. A closed-ended resource hold that is not completed within a fixed duration after it was requested will expire. Only a researcher patron can request an open-ended resource hold. Any patron can set a preference, rejecting holds on resources with usage fees. Any patron with more than two overdue checkouts at a library branch cannot request a hold at that same library branch. See Figure 4.15.

15. See p. 85, Rule 65.
16. See p. 82, Rule 59.

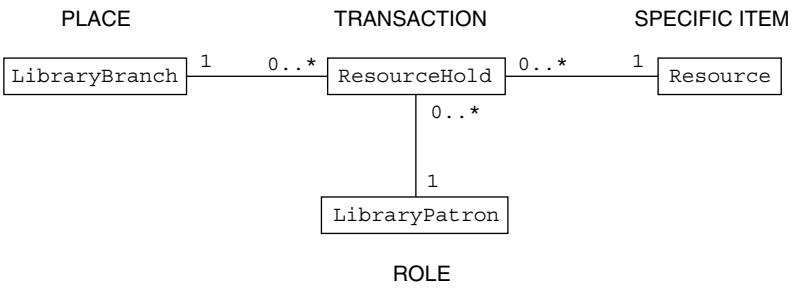

FIGURE 4.15
A LibraryPatron can request holds for library Resources at LibraryBranches.

Collaboration Rules

Type
R48. [Role] A regular library patron can only collaborate with closed-ended resource holds; a researcher patron can collaborate with open-ended or closed-ended resource holds.

Multiplicity
R49. [Transaction] A resource hold knows exactly one library patron, and cannot remove it.

R50. [Role] A library patron knows zero to many resource holds. If the patron is an educational researcher, he can collaborate with an unlimited number of resource holds; otherwise, the patron has an upper limit, currently five, to the number of resource holds per month.

Property
R51. [Role] To collaborate with a resource hold, a library patron must be an adult with a valid library registration number.

State
R52. [Role] A library patron whose registration has expired or has been inactivated cannot collaborate with a resource hold.

Conflict
R53. [Role] A library patron with a preference against resources with fees cannot collaborate with a resource hold for a resource with a fee, unless the patron has indicated that preferences should be ignored for this resource hold (here, the `role` conflicts with the `specific item`).

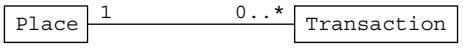

FIGURE 4.16
The `transaction - place` pattern.

Transaction – Place

Transaction – Place Collaboration Rules

The `transaction - place` collaboration is another pattern representing an entity participating in a `transaction`. These patterns use collaboration rules to define the entity's requirements for participating in the `transaction`. A `place` represents a location where interactions happen, and it knows its surroundings and their limitations. Therefore, a `place` is better qualified to judge whether an interaction should occur there. To personify the `place`, it has a strong vested interest in deciding what kind of people and things can show up at its location, and what they are permitted to do. Conclusion: A `place` does most of the work of checking its requirements for a `transaction`, as shown in Table 4.9.

TABLE 4.9
`Transaction - Place` Collaboration Rules

	Transaction	Place
Type		✓
Multiplicity	✓	✓
Property		✓
State		✓
Conflict		✓

Type Rules

RULE 54 A `place` knows what kinds of `transactions` it can accept.

Multiplicity Rules

RULE 55 A `transaction` knows about one `place` and cannot break with it.

RULE 56 A `place` knows zero to many `transactions`; typically, a `place` cannot remove a `transaction`.

A `transaction` generally happens at a `place`. If that information is recorded, it does not make real-world sense to forget where the `transaction` happened while the `transaction` remains in the system.

Property Rules

RULE 57 A `place` can require that `transactions` have properties within certain ranges or conform to certain standards.

The `place` decides if the `transaction` is in any other way beyond the limits of the `place`.

State Rules

RULE 58 A `place` checks its state before accepting a `transaction`.

Closed, under repair, or full `places` cannot accept new `transactions`.

Conflict Rules

RULE 59 A `place` checks for conflicts between its existing `transactions` and a new `transaction`. A `place` also checks for conflicts between itself and the other collaborations of a `transaction`.

A `place` has the responsibility of deciding if it can handle overlapping `transactions`, and so owns the conflict rule for coinciding `transactions`. A `place` can also refuse a `transaction` if its `specific item` or `role` is not suitable for the `place`; this occurs when the `place` has filters for screening out undesirable `specific items` or `roles`.

Transaction – Place Collaboration Rules Example

EXAMPLE—In the library resource example, some library branches do not allow open-ended resource holds, and some do not allow any resource holds except by researcher patrons. A library branch will not accept any resource holds while it is undergoing its yearly inventory review. A few library branches in religious communities have non-working hours during which resource holds cannot be requested. See Figure 4.15.

Collaboration Rules

Type

R54. [Place] A library branch knows the types of resource holds it allows—open-ended, closed-ended, or both.

Multiplicity

R55. [Transaction] A resource hold happens at exactly one place, where the resource is located.

R56. [Place] A library branch can collaborate with an unlimited number of resource holds.

Property

R57. [Place] A library branch with non-working hours cannot collaborate with a resource hold whose timestamp occurs during those hours.

State

R58. [Place] A library branch checks its state before accepting a resource hold; it cannot accept a hold while its inventory review is in progress.

Conflict

R59. [Place] A library branch that only allows resource holds by researcher patrons cannot accept a hold from a regular library patron (here, the place conflicts with the role).

Transaction – Specific Item

Transaction – Specific Item Collaboration Rules

The `transaction - specific item` collaboration also represents an entity participating in a `transaction`. A `specific item` represents a thing that can participate in interactions; it knows its properties and restrictions, and is qualified to judge if it belongs in an interaction. As with the other entities involved with `transactions`, the `specific item` does most of the work of checking its requirements for a `transaction`, as shown in Table 4.10.

FIGURE 4.17
The `transaction - specific item` pattern.

TABLE 4.10
`Transaction` - `Specific Item` Collaboration Rules

	Transaction	Specific Item
Type		✓
Multiplicity	✓	✓
Property		✓
State		✓
Conflict		✓

Type Rules

RULE 60 A `specific item` knows what types of `transactions` it can accept.

Multiplicity Rules

RULE 61 A `transaction` knows about one `specific item` and cannot break with it.

RULE 62 A `specific item` knows zero to many `transactions`; typically, a `specific item` cannot remove a `transaction`.

A `transaction's` sole purpose is to record a `specific item's` interaction. `Specific items` exist to a great extent for the purpose of taking part in a `transaction`.

Property Rules

RULE 63 A `specific item` can require that `transactions` have properties within certain ranges or conform to certain standards.

The `specific item` decides if the `transaction` does not fit within the `specific item's` limits.

State Rules

RULE 64 A `specific item` checks its state before accepting a `transaction`.

Out-of-stock, discontinued, or defective `specific items` cannot accept new `transactions`.

Conflict Rules

RULE 65 A `specific item` checks for conflicts between its existing `transactions` and a new `transaction`. A `specific item` also checks for conflicts between itself and the other collaborations of a `transaction`.

A `specific item` has the responsibility of deciding if it can handle overlapping `transactions`, and so owns the conflict rule for coinciding `transactions`. A `specific item` can also refuse a `transaction` if its `place` or `role` is not suitable for the `specific item`; this occurs when the `specific item` has filters for screening out undesirable `places` or `roles`.

Transaction – Specific Item Collaboration Rules Example

EXAMPLE—In the library example, a resource can specify if it requires a closed-end resource hold; typically, such a resource needs special storage or is very popular. A resource can include retrieval fees, which are necessary to bring it to the library branch from special storage. These fees are collected when the resource hold is requested. Resource usage fees are charged during checkout, and cover the costs of viewing the resource. See Figure 4.15.

Collaboration Rules

Type

R60. [Specific Item] A resource knows the type of resource holds it can accept—open-ended, closed-ended, or both.

Multiplicity

R61. [Transaction] A resource hold knows exactly one resource; if the resource hold removes its resource, the hold is no longer valid.

R62. [Specific Item] A resource knows zero to many resource holds.

Property

R63. [Specific Item] A resource with a retrieval fee cannot collaborate with a resource hold that has not collected adequate payment to cover the fee.

State

R64. [Specific Item] A damaged or defective resource cannot collaborate with a resource hold.

Conflict
R65. [Specific Item] A restricted resource cannot collaborate with a resource hold by a regular library patron. A resource cannot collaborate with a resource hold if the resource already has an active resource hold.

Aggregate Event Collaboration Rules

Distributing Collaboration Rules

Unlike events that involve only one person, one place, and one thing, aggregate events involve multiple people, places, or things. To distinguish the separate interactions within it, an aggregate event is decomposed into smaller constituent events that record the details of a single entity's involvement in the interaction. Because these constituent events are the by-products of requesting a collaboration between an entity and the aggregate event, their fates rest not on their own collaboration rules, but on the collaboration rules between the aggregate event and the participating entity. If the entity cannot participate in the aggregate event, then the constituent event cannot either. Likewise, a constituent event can be removed only if the entity it represents can be removed.

Details vs. Collaboration Rules

Often when an entity seeks to participate in an aggregate event, it presents details about its interaction along with its request, for example, quantities involved, sales commissions, or time durations. Verifying these details is not the responsibility of the collaboration rules, which only determine if the entity is qualified to participate; instead, the constituent event created to represent the entity's involvement verifies the details, which are represented as its properties. Should one or more properties be invalid, then appropriate action can be taken.

EXAMPLE—A customer uses an online system to order given quantities of products from a local office supply store. If a larger quantity of product is requested than is in stock, the product is still placed in the order. Store policies and customer preferences determine whether the order is held until more stock arrives, or if it is broken into multiple orders.

Composite Transaction – Line Item

Composite Transaction – Line Item Collaboration Rules

The composite transaction - line item collaboration represents the involvement of many entities in an aggregate event. As a constituent event within the composite transaction, a line item captures the interaction details of a single entity. Whether the line item collaborates with the composite transaction depends mostly on the collaboration rules between the composite transaction and the line item's entity; however, there are a few collaboration rules between the two that are largely technical rules enforcing the meaning of the collaboration. The line item assumes most of the responsibility for these few rules because as the one adding further details to the aggregate event, it can be considered more particular or detailed than the composite transaction. This distribution of work is shown in Table 4.11.

TABLE 4.11
Composite Transaction - Line Item Collaboration Rules

	Composite Transaction	Line Item
Type		✓
Multiplicity	✓	✓
Property		
State		
Conflict		

Type Rules

RULE 66 A line item collaborates with only one type of composite transaction.

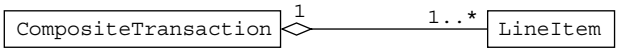

FIGURE 4.18
The composite transaction - line item pattern.

The composite transaction is the object factory for creating line items; as the most detailed, the line item has responsibility for ensuring it is not added to the wrong type of composite transaction.[17]

Multiplicity Rules

RULE 67 A composite transaction knows about at least one line item.

RULE 68 A line item knows exactly one composite transaction; a line item removed from its composite transaction is invalid.

A composite transaction without line items is an interaction without things interacting, which is not useful. Transferring line items to another composite transaction is not allowed.

Composite Transaction – Line Item Collaboration Rules Example

EXAMPLE—A customer uses an online system to order given quantities of products for delivery from, or pick-up at, a local office supply store. For each product ordered, a sales order line item captures its quantity ordered and allows it to be adjusted prior to submission of the order. See Figure 4.19.

Collaboration Rules

Type
R66. [Line Item] A sales order line item can only collaborate with a sales order.

Multiplicity
R67. [Composite Transaction] A sales order must include at least one sales order line item.

R68. [Line Item] A sales order line item knows exactly one sales order; it cannot be transferred to another, and if removed, it becomes invalid.

17. This responsibility can be as simple as the parameter type in a setter method; for untyped languages, more work is required.

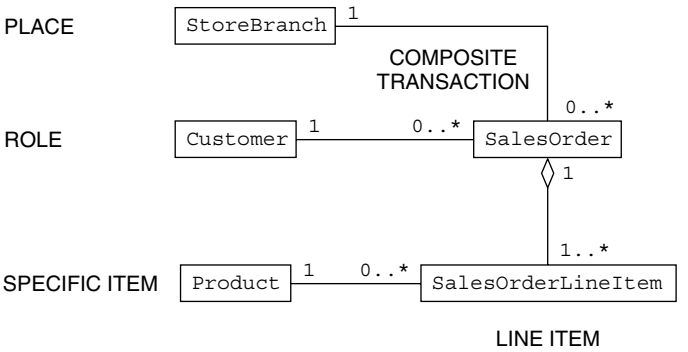

FIGURE 4.19
A SalesOrder creates SalesOrderLineItems for Products included in it.

Specific Item – Line Item

Specific Item – Line Item Collaboration Rules

The `specific item - line item` collaboration records the participation of a single entity in an aggregate event. While it is tempting to look at this like a `transaction - specific item`, that does not work because the `specific item` is not really aware of the `line item`, as it is with a `transaction`.[18] For an aggregate event, the `specific item` has collaboration rules for the `composite transaction` and conflict rules for its collaborators. Between the `line item` and `specific item`, collaboration rules are mere bookkeeping. In fact, there is not even a type rule. If the `specific item` is the right type for the `composite transaction`, then it is the right type for the `line item`. As shown in Table 4.12, multiplicity rules are the only collaboration rules checked in this pattern.

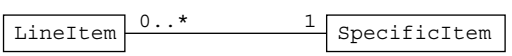

FIGURE 4.20
The `specific item - line item` pattern.

18. In Java implementations of this pattern, the `line item` implements an interface that gives it the appearance of a `composite transaction`, thus fooling the `specific item` into believing it is collaborating directly with the `composite transaction`.

TABLE 4.12
`Specific Item - Line Item` Collaboration Rules

	Line Item	Specific Item
Type		
Multiplicity	✓	✓
Property		
State		
Conflict		

Multiplicity Rules

RULE 69 A `line item` collaborates with exactly one `specific item`.

RULE 70 A `specific item` knows about zero to many `line items`.

The `specific item` relates to the `line items` as if it they were `composite transactions`. Because the `line items` represent the `specific item's` history of interactions, the `specific item` cannot remove them.

Specific Item – Line Item Collaboration Rules Example

EXAMPLE—A customer uses an online system to order given quantities of products for delivery from or pick-up at a local office supply store. Certain products are not available for delivery orders, and not all store branches carry all the products in the online catalog. Out-of-stock products are included in the order, but delivered or available for pick-up later. Specialty and promotional products are available only for preferred customers whose total yearly purchases have exceeded a minimum dollar amount. See Figure 4.19.

Collaboration Rules

Multiplicity
R69. [Line Item] A sales order line item knows exactly one product; it cannot remove it.

R70. [Specific Item] A product knows zero to many sales order line items; it may know them as sales orders. The product cannot remove any sales order line items.

Conflict Collaboration Rules

These event collaboration rules are run when a product is added to a sales order. Should any of them fail then the product is not added, and no sales order line item is created.

Type

R60. [Specific Item] A product knows the type of sales order it can accept—delivery or pick-up.

Conflict

R65. [Specific Item] A specialty or promotional product cannot collaborate with a sales order from a customer who is not preferred. A product cannot collaborate with a sales order from a store branch that does not stock it.

Transaction – Follow-up Transaction

Transaction - Follow-up Transaction Collaboration Rules

In the `transaction - follow-up transaction` collaboration, the `transaction` is acting more like a specific entity than an event. The `transaction` is now the participant in a new interaction represented by the `follow-up transaction`. As the one interacting, the `transaction` has ownership of the collaboration rules, as shown in Table 4.13.

Type Rules

RULE 71 A `transaction` knows what types of `follow-up transactions` can follow it.

Multiplicity Rules

RULE 72 A `follow-up transaction` knows about one `transaction` and cannot break with it.

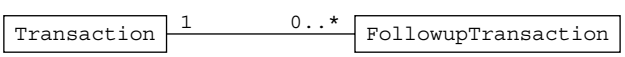

FIGURE 4.21
The `transaction - follow-up transaction` pattern.

TABLE 4.13
`Transaction` – `Follow-up Transaction` Collaboration Rules

	Transaction	Follow-up Transaction
Type	✓	
Multiplicity	✓	✓
Property	✓	
State	✓	
Conflict	✓	

RULE 73 A `transaction` knows zero to many `follow-up transactions`; a `transaction` cannot remove a `follow-up transaction`.

Property Rules

RULE 74 A `transaction` can require that `follow-up transactions` have properties within certain ranges or conform to certain standards.

The `transaction` decides if the `follow-up transaction` does not meet the `transaction's` requirements.

State Rules

RULE 75 A `transaction` checks its state before accepting a `follow-up transaction`.

Conflict Rules

RULE 76 A `transaction` checks for conflicts between its existing `follow-up transactions` and a new `follow-up transaction`. A `transaction` also checks for conflicts between itself and the other collaborations of a `follow-up transaction`.

Transaction – Follow-up Transaction Collaboration Rules Example

EXAMPLE—In the office supply example, a sales order is followed by some number of deliveries. A sales order that is cancelled or incom-

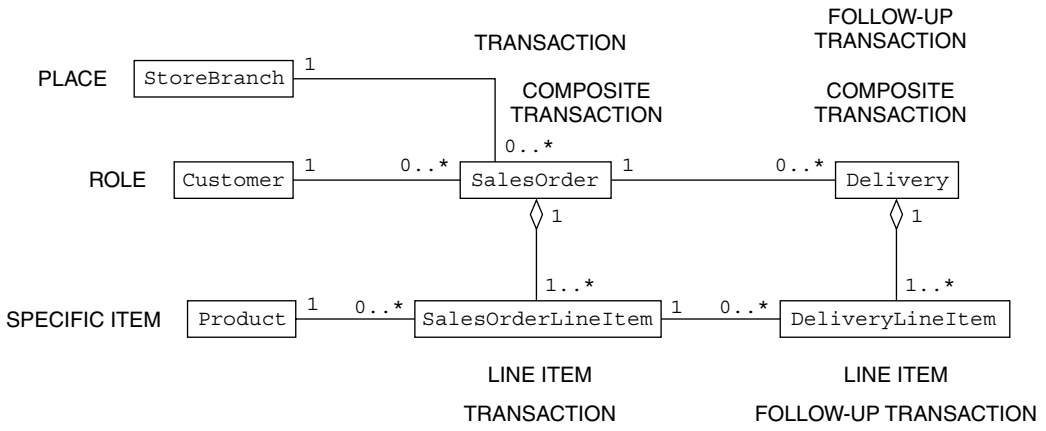

FIGURE 4.22
An online SalesOrder is followed by Deliveries.

plete cannot accept any deliveries. A sales order marked for hold until complete cannot be followed by a partial delivery. See Figure 4.22.

Collaboration Rules

Type
R71. [Transaction] A sales order knows the type of delivery that can follow it—partial or complete.

Multiplicity
R72. [Follow-up Transaction] A delivery knows exactly one sales order, and cannot break with it.

R73. [Transaction] A sales order knows zero to many deliveries, and cannot remove them, although a delivery can be cancelled.

Property
R74. [Transaction] A sales order rejects any delivery that does not have the same shipping address as itself.

State
R75. [Transaction] A sales order that is incomplete or cancelled cannot accept a delivery.

Conflict
R76. [Transaction] A sales order cannot accept a delivery involving sales order line items that have already been completely delivered.