2. Modeling with UML: Solutions

2-1 Consider an ATM system. Identify at least three different actors that interact with this system.

An actor is any entity (user or system) that interacts with the system of interest. For an ATM, this includes:

- Bank Customer
- ATM Maintainer
- Central Bank Computer
- Thief

The last actor is often referred to as a "misactor" in the literature, because it is an actor that interacts with the system but shouldn't.

2–2 Can the system under consideration be represented as an actor? Justify your answer.

The system under consideration is not external to the system and shouldn't be represented as an actor. There are a few cases, however, when representing the system as an actor may clarify the use case model. These include situations where the system initiates uses cases, for example, as time passes (Check for Outdated Articles, Send Daily Newsletter).

2–3 What is the difference between a scenario and a use case? When do you use each construct?

A scenario is an actual sequence of interactions (i.e., an instance) describing one specific situation; a use case is a general sequence of interactions (i.e., a class) describing all possible scenarios associated with a situation. Scenarios are used as examples and for clarifying details with the client. Use cases are used as complete descriptions to specify a user task or a set of related system features.

2–4 Draw a use case diagram for a ticket distributor for a train system. The system includes two actors: a traveler, who purchases different types of tickets, and a central computer system, which maintains a reference database for the tariff. Use cases should include: `BuyOneWayTicket`, `BuyWeeklyCard`, `BuyMonthlyCard`, `UpdateTariff`. *Also include the following exceptional cases:* `Time-Out` *(i.e., traveler took too long to insert the right amount),* `TransactionAborted` *(i.e., traveler selected the cancel button without completing the transaction),* `DistributorOutOfChange`, *and* `DistributorOutOfPaper`.

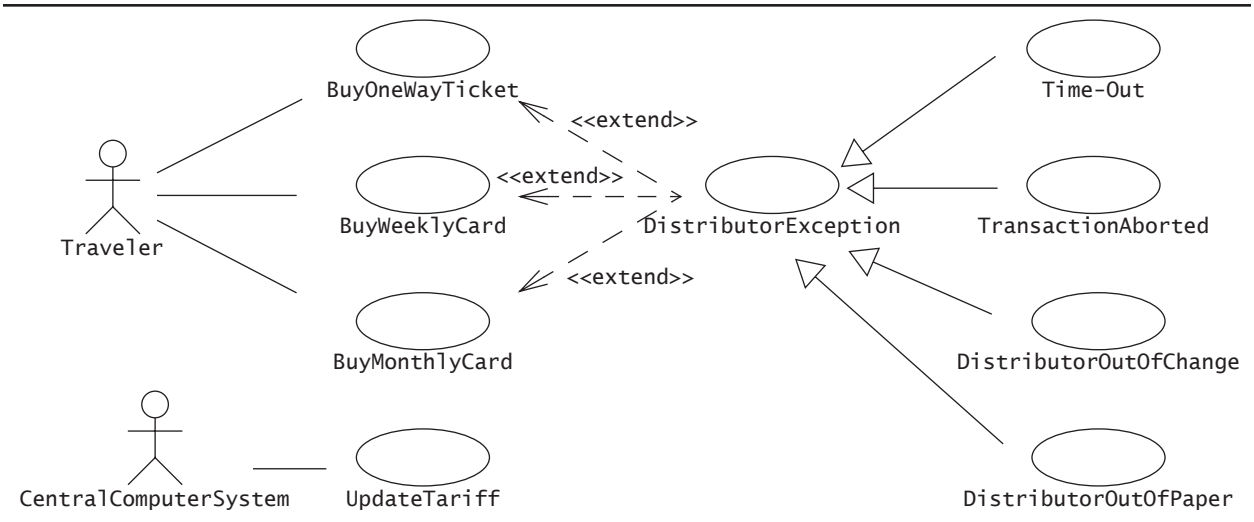

Figure 2-1 Example solution to Exercise 2–4.

This questions can have several correct answers, Figure 2-1 being a possible answer. The following elements should be present:

- The relationship between an actor and a use case is a communication relationship (undirected solid line).
- The relationship between exceptional use cases and common use cases is an `<<extend>>` relationship.
- The exceptional use cases described in the exercise only apply to the use cases invoked by the traveler.

The following elements should be present in a "good" answer:

- All exceptions apply to all traveler use cases. Instead of drawing 3x4 relationships between these use cases, an abstract use case from which the exceptional use case inherit can be used, thus reducing the number of `<<extend>>` relationships to 3 at the cost of introducing 4 generalization relationships.
- The student can introduce exceptional use cases not specified in the exercise that apply to the `CentralComputerSystem` use cases.

2–5 Write the flow of events and specify all fields for the use case UpdateTariff that you drew in Exercise 2–4. Do not forget to specify any relationships.

Figure 2-2 depicts a possible solution for this exercise.

Use case name	`UpdateTariff`
Participating actor	Initiated by `CentralComputerSystem`
Flow of events	1. The `CentralComputerSystem` activates the "UpdateTariff" function of the ticket distributors available on the network. 2. The ticket distributor disables the traveler interface and posts a sign indicating that the ticket distributor is under maintenance. 3. The ticket distributor waits for the new database from the `CentralComputerSystem`. 4. After waiting a minute for the ticket distributors to reach a waiting state, the `CentralComputerSystem` broadcasts the new database. 5. The ticket distributor system receives the new database of tariff. Upon complete, the ticket distributor sends an acknowledgement to the `CentralComputerSystem`. 6. After acknowledgment, the ticket distributor enables the traveler interface and can issue tickets at the new tariff. 7. The `CentralComputerSystem` checks if all ticket distributors have acknowledged the new database. If not, the CentralComputerSystem invokes the `CheckNonRespondingDistributors` use case.
Entry condition	• The ticket distributor is connected to a network reachable by the CentralComputerSystem.
Exit condition	• The ticket distributor can issue tickets under the new tariff, OR • The ticket distributor is disabled and displays a sign denoting that it is under maintenance.
Quality requirements	• The ticket distributor stays offline at most 2 minutes and is considered out-of-order otherwise.

Figure 2-2 A possible solution for the `UpdateTariff` use case.

2–6 Draw a class diagram representing a book defined by the following statement: "A book is composed of a number of parts, which in turn are composed of a number of chapters. Chapters are composed of sections." Focus only on classes and relationships.

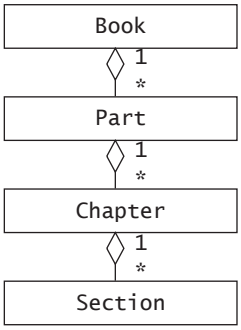

Figure 2-3 Example solution for Exercise 2–6.

This exercise checks the student's understanding of basic aspects of class diagrams, including:

- Classes are represented with rectangles.
- The attribute and operations compartment can be omitted.
- Aggregation relationships are represented with diamonds.
- Class names start with a capital letter and are singular.

2–7 Add multiplicity to the class diagram you produced in Exercise 2–6.

See Figure 2-3. Aggregation does not imply multiplicity, thus the `1..*` multiplicity is necessary.

2–8 Draw an object diagram representing the first part of this book (i.e., Part I, Getting Started). Make sure that the object diagram you draw is consistent with the class diagram of Exercise 2–6.

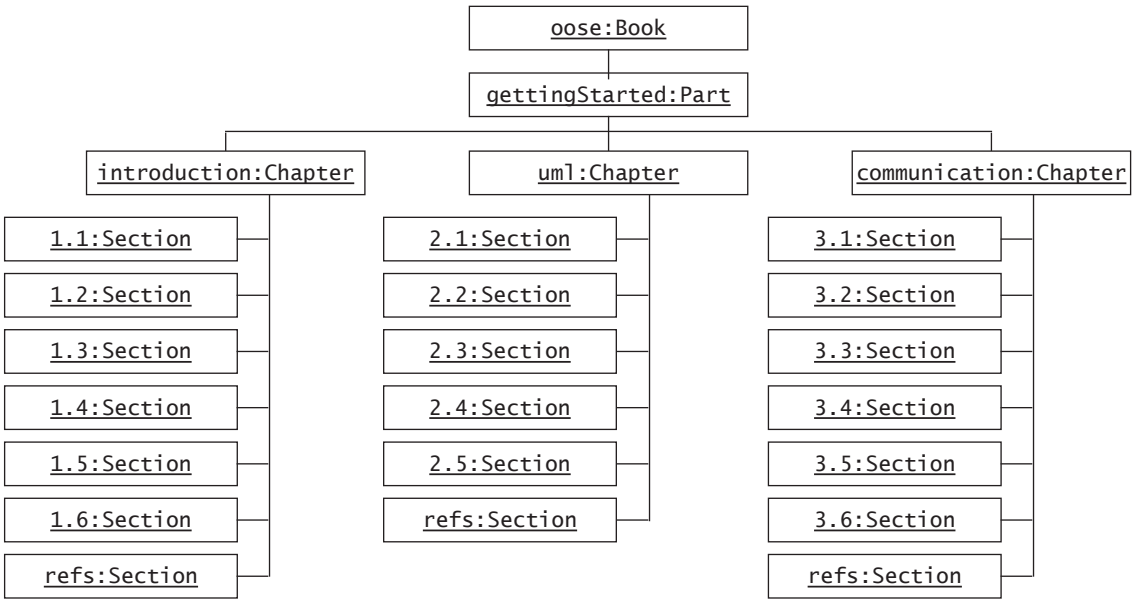

Figure 2-4 Example solution for Exercise 2–8.

This exercise checks the student's understanding of basic aspects of object diagrams, including:

- Objects are represented with rectangles and underlined labels.
- The class of an object is included in the label of the object (e.g., `uml:Chapter` is of class `Chapter`).
- Links are represented with solid lines

2–9 Extend the class diagram of Exercise 2–6 to include the following attributes:

- *a book includes a publisher, publication date, and an ISBN*
- *a part includes a title and a number*
- *a chapter includes a title, a number, and an abstract*
- *a section includes a title and a number*

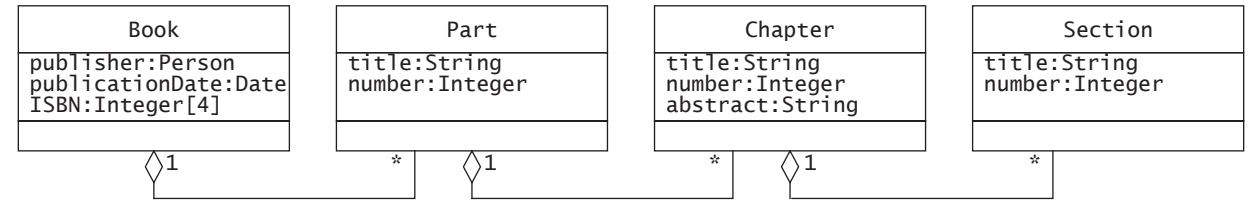

Figure 2-5 Example solution for Exercise 2–9.

This exercises checks the student's knowledge of attributes and their representation in UML (page 45).

2–10 Consider the class diagram of Exercise 2–9. Note that the `Part`, `Chapter`, *and* `Section` *classes all include a title and a number attribute. Add an abstract class and a generalization relationship to factor out these two attributes into the abstract class.*

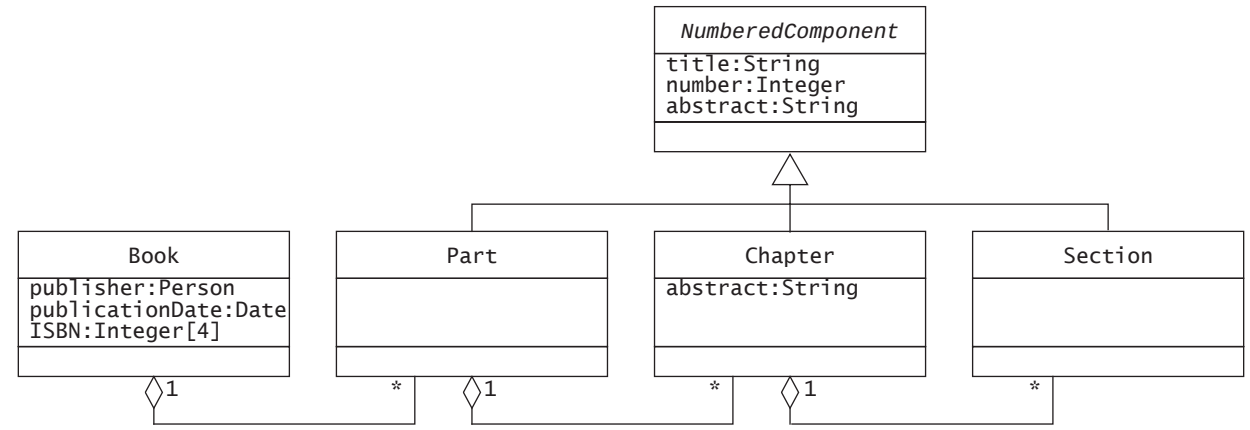

Figure 2-6 Example solution for Exercise 2–10.

This exercise checks the student's knowledge of abstract classes and inheritance.

2–11 Draw a class diagram representing the relationship between parents and children. Take into account that a person can have both a parent and a child. Annotate associations with roles and multiplicities.

Figure 2-7 depicts a canonical solution. This exercise is meant to emphasize the difference between a relationship, a role, and a class. In the above sentence, parent and child are roles while person is the class under consideration. This results in a class diagram with a single class and a single association with both ends to the class.

A common modeling mistake by novices is to draw two classes, one for the parent and one for the child.

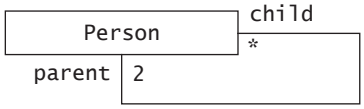

Figure 2-7 Example solution for Exercise 2–11.

2–12 Draw a class diagram for bibliographic references. Use the references in Appendix C, Bibliography, to test your class diagram. Your class diagram should be as detailed as possible.

The domain of bibliographic references is rich and complex. Consequently, students should deepened their understanding of the domain before they attempt to draw a class diagram (similar to the requirements analysis of a system). Figure 2-8 depicts an incomplete sample solution that could be accepted as sufficient from an instructor. The class diagram should minimally include the following concepts:

- An abstract class *`Publication`* (or *`BibliographicReference`*)
- A many to many relationship between `Author` and *`Publication`*
- At least three or more concrete classes refining *`Publication`*
- At least one aggregation relationship (e.g., between `Journal` and `Article` or `Proceedings` and `ConferencePaper`). Both ends of the aggregation should also be subclasses of *`Publication`*.

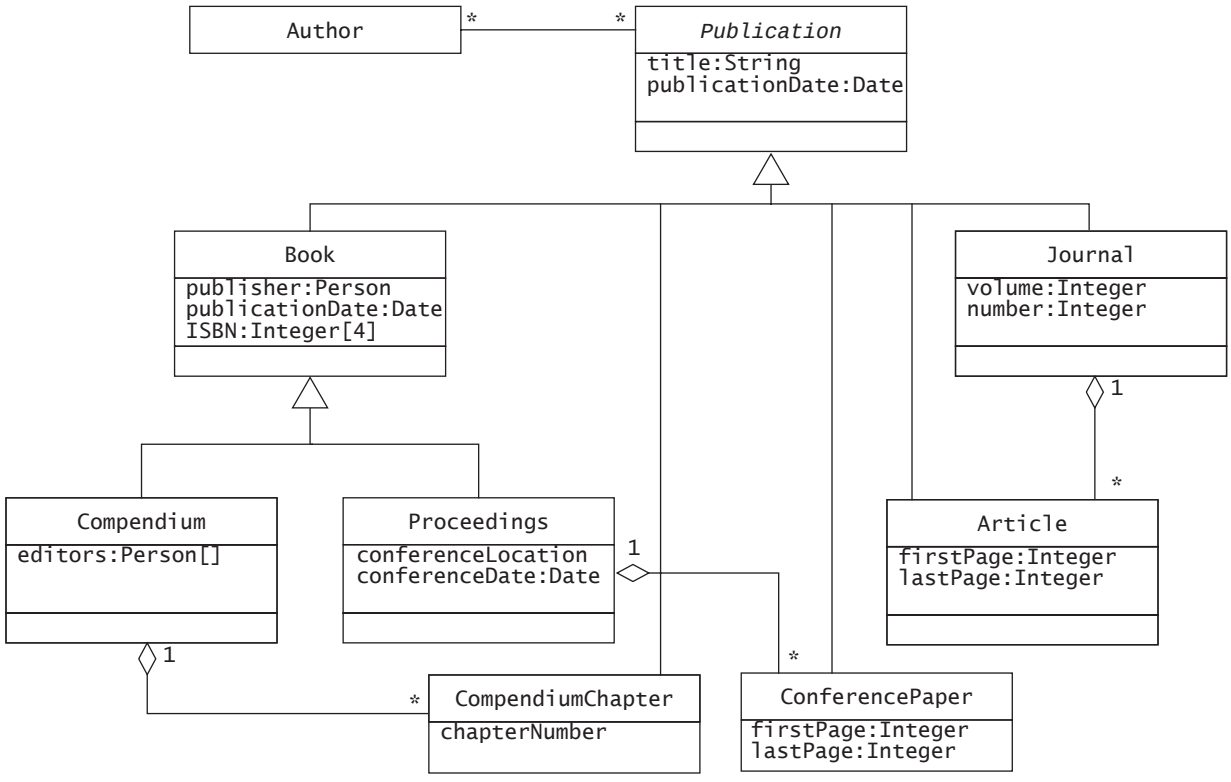

Figure 2-8 Example solution for Exercise 2–12.

2–13 Draw a sequence diagram for the `warehouseOnFire` *scenario of Figure 2-15. Include the objects* `bob`, `alice`, `john`, `FRIEND`, *and instances of other classes you may need. Draw only the first five message sends.*

This exercise checks the student's knowledge of sequence diagrams when using instances. This exercise can have several correct answers. In addition to the UML rules on sequence diagrams, all correct sequence diagrams for this exercise should include one or more actors on the left of the diagram who initiate the scenario, one or more objects in the center of the diagram which represent the system, and a dispatcher actor on the right of the diagram who is notified of the emergency. All actors and objects should be instances. Figure 2-9 depicts a possible answer.

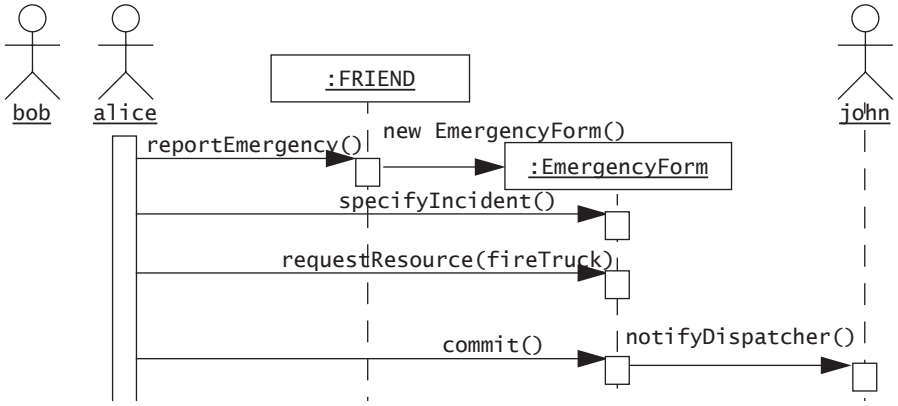

Figure 2-9 Example solution for Exercise 2–13.

2–14 Draw a sequence diagram for the `ReportIncident` use case of Figure 2-14. Draw only the first five message sends. Make sure it is consistent with the sequence diagram of Exercise 2–13.

This exercise tests the student's knowledge of sequence diagram when using classes. Like the previous exercise, this exercise can have several correct solutions. A correct solution should include a FieldOfficer actor on the left, a Dispatcher actor on the right, and one or more classes in the middle representing the FRIEND system. The answer to this exercise should be consistent with the answer to the previous exercise, in the sense that all class and operation names should be the same. Figure 2-10 depicts a possible answer.

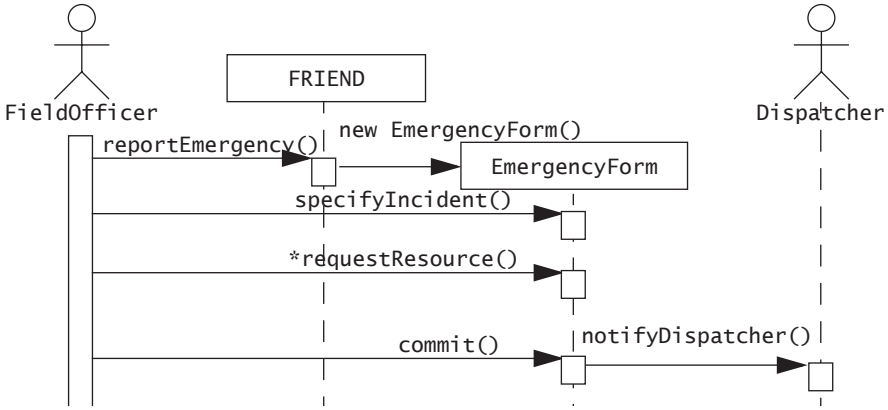

Figure 2-10 Example solution for Exercise 2–14.

2–15 Consider the process of ordering a pizza over the phone. Draw an activity diagram representing each step of the process, from the moment you pick up the phone to the point where you start eating the pizza. Do not represent any exceptions. Include activities that others need to perform.

Figure 2-11 is an example solution for this exercise. The following elements should be present in the solution:

- Activity names should be verb phrases indicating what the initiating actor is attempting to accomplish. Roles should be indicated with swimlanes

- Activities that are concurrent or which do not need to happen in a sequential order should be indicated with complex transitions.

2–16 Add exception handling to the activity diagram you developed in Exercise 2–15. Consider at least three exceptions (e.g., delivery person wrote down wrong address, deliver person brings wrong pizza, store out of anchovies).

This exercise checks the student's knowledge of decision points. Figure 2-12 depicts an example solution. Students modeling a realistic example will also think about cascaded exceptions and learn to represent them with multiple decision points.

2–17 Consider the software development activities which we described in Section 1.4. Draw an activity diagram depicting these activities, assuming they are executed strictly sequentially. Draw a second activity diagram depicting the same activities occurring incrementally (i.e., one part of the system is analyzed, designed, implemented, and tested completely before the next part of the system is developed). Draw a third activity diagram depicting the same activities occurring concurrently.

This exercise tests the student's knowledge of the activity diagram syntax, not the knowledge of software engineering activities. In the sample solution provided in Figure 2-13, we assumed that requirements elicitation need to be completed before any subsystem decomposition can be done. This leads to a splitting of the flow of control in the third diagram where all remaining activities occur concurrently. The instructor could accept a solution where requirements elicitation also occurs incrementally or concurrently with the other development activities.

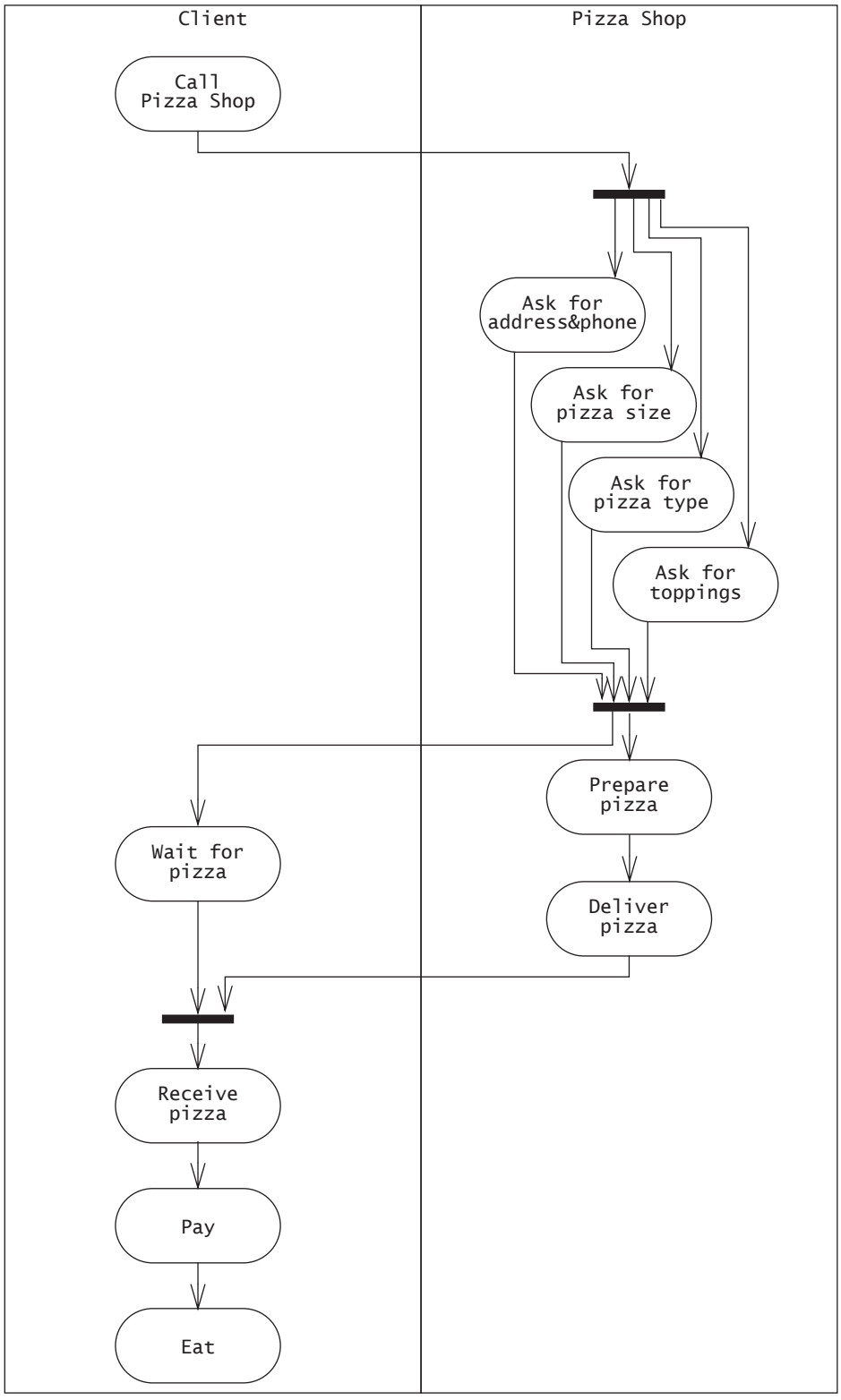

Figure 2-11 An example of pizza order activity diagram with exception handling.

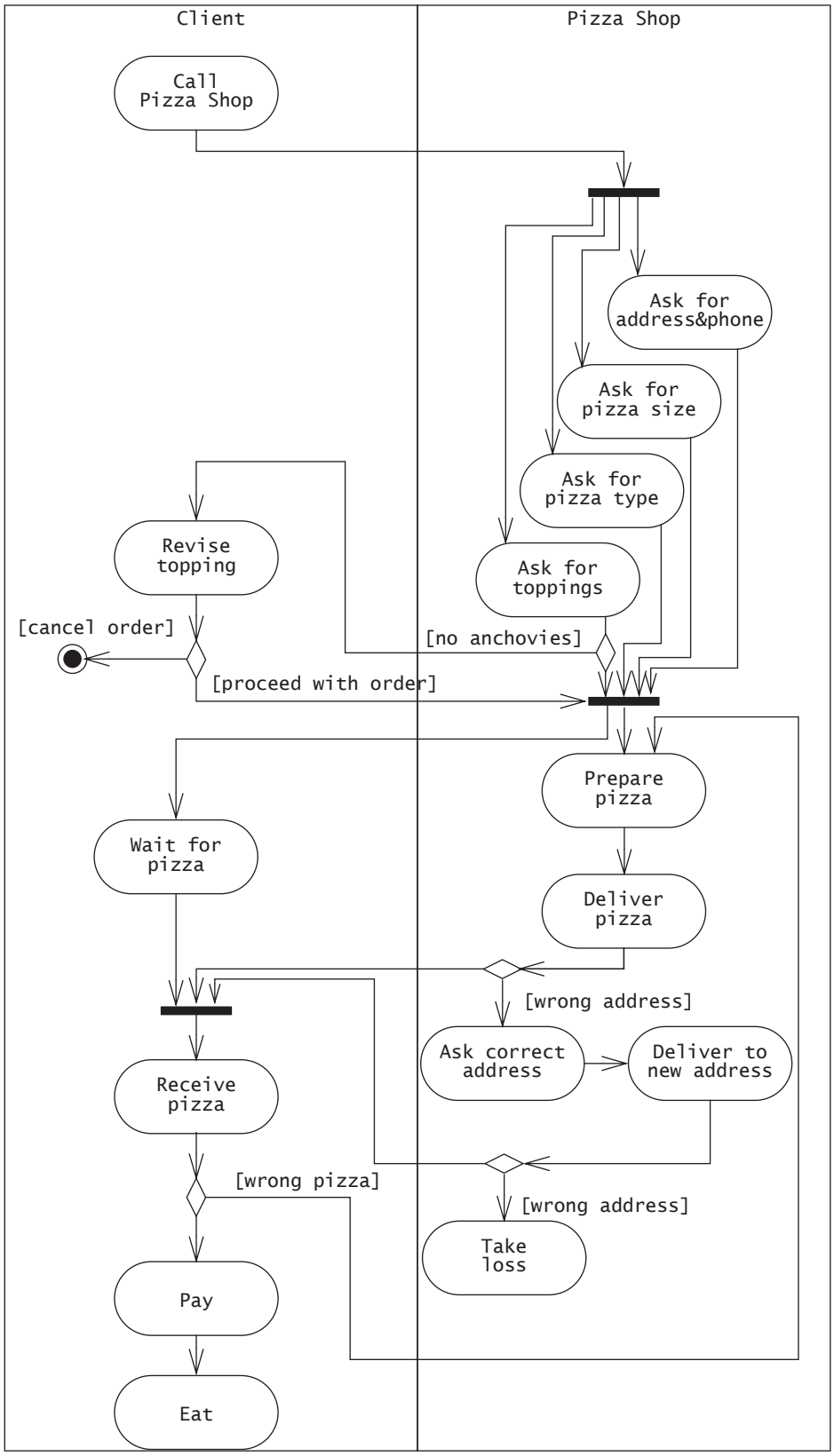

Figure 2-12 An example of pizza order activity diagram.

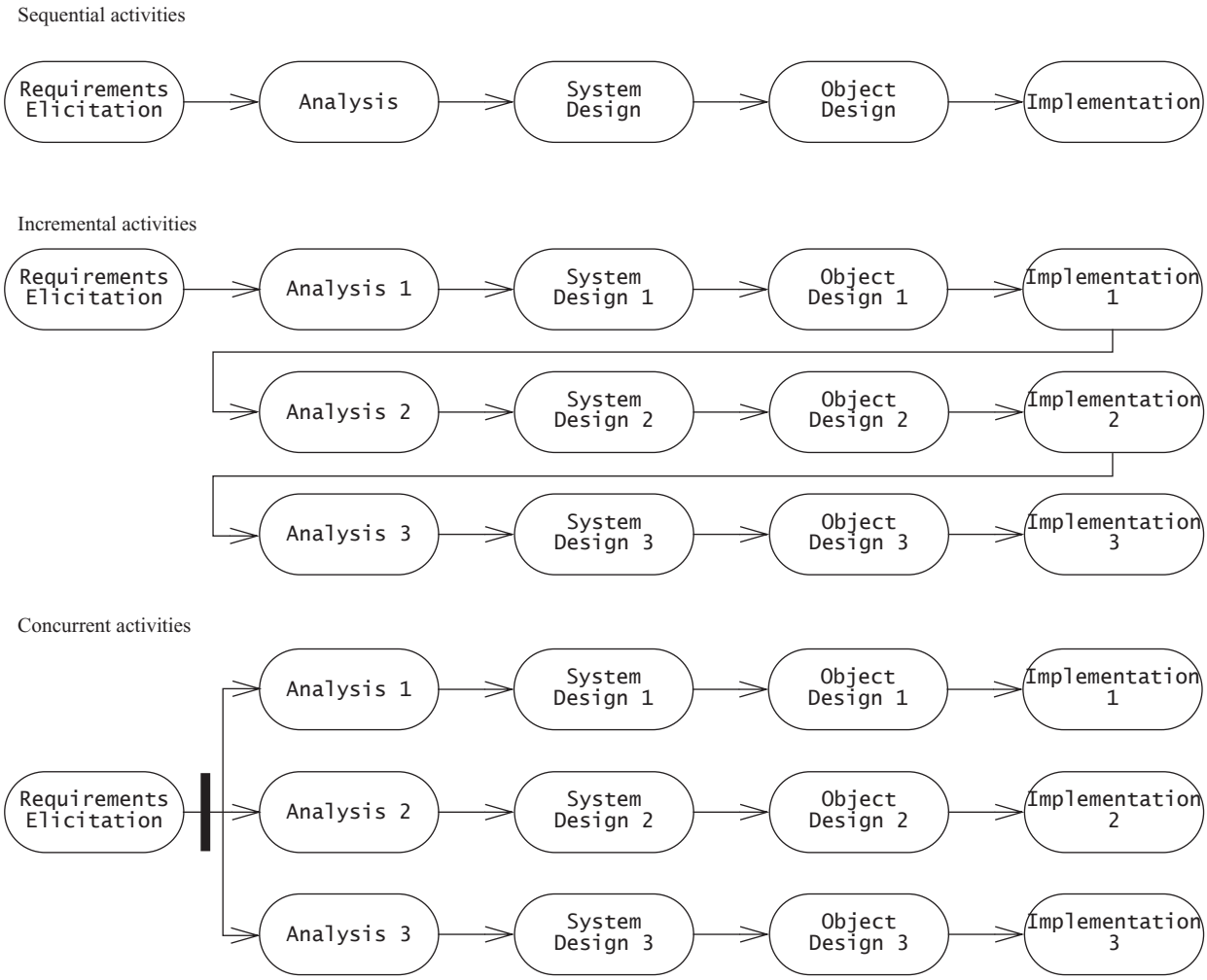

Figure 2-13 Example solution for Exercise 2–17.

