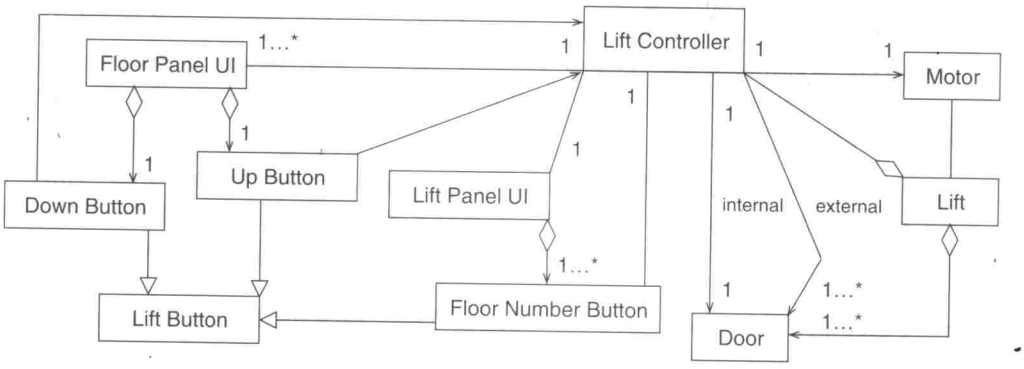
- Map conditional branchings to conditional statements such as if-else statements
- Implement active objects using threads
- Map concurrent branching using concurrent threads

Case Study: A Lift Control System

In previous sections, the basic techniques for implementing individual UML diagrams were discussed. Now, we will illustrate how all these techniques can be applied to a real-life problem. The following is a description of a simplified lift control system.

A lift consists of a door, a motor and a lift controller. The lift controller is responsible for controlling the lift system (see the class diagram in Figure 5.48). Passengers interact with the lift system by pressing buttons on individual floors or on the control panel inside the lift. Normally, the lift stays on the ground floor (*0*-th floor) of a building. If a passenger enters the lift and presses the button for the k-th floor, the lift will rise to the k-th floor. When the lift arrives at requested floor (say k-th floor), it opens the door for M seconds and then closes it. The lift then becomes idle.

Figure 5.48. Analysis class diagram of lift control system



A passenger on m-th floor calls a lift by pressing the *up* or *down* button. The lift will move to the m-th floor and open the door on arrival. The passenger requests to go to a particular floor by pressing the corresponding button on the control panel inside the lift. If there is no passenger interaction on the control panel within M seconds, then the lift will return to the ground floor.

With the analysis class diagram and the problem statement, we will be able to develop sequence diagrams to further elaborate the dynamic behaviors of the system in different scenarios. The sequence diagrams can then be used to develop the state diagram of the lift controller, which is the control object of the whole system.

The following is the first scenario of the lift control system.

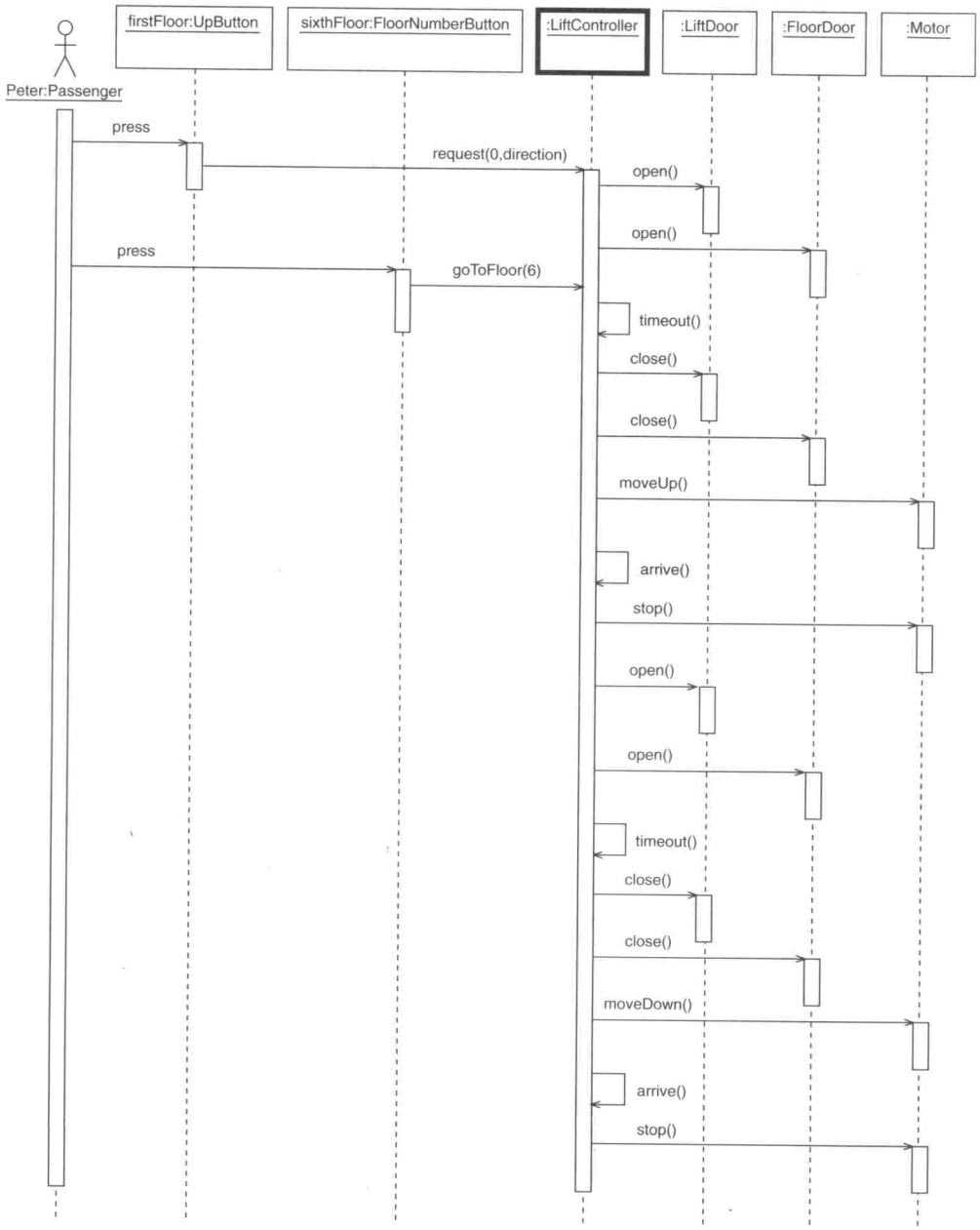
Scenario 1

A passenger, Peter, walks into the lift lobby on the ground floor of the building. He presses the UP button and waits for the lift's arrival. On arrival, the lift opens, he enters and presses the sixth floor button on the control panel inside the lift. The lift closes and goes up until it arrives on the sixth floor. The lift opens and Peter walks out of the lift. The lift waits for a short while (ten seconds), closes and then goes down to the ground floor. The lift will stay at the ground floor until further user interaction.

The sequence diagram for the first scenario is shown in Figure 5.49. It is developed by going through each step of the sequence of transactions of the scenario. For each step, the actor performs an action and the system responds accordingly. The actor's input and system's response will help to determine what internal actions are required inside the system. The actor's input is received by the boundary object(s). The boundary object will then send a message to the control object to handle the actor's input. Then the control object needs to send one or more messages to the entity object(s) to perform the actual required actions. Finally, the control object will send message(s) to the boundary object(s) to respond to the actor. The following is the sequence of actions for the scenario:

- Peter presses the UP button on the ground floor of the building. The press button event is sent to the lift controller. Now, consider what the lift system would do. Since the lift is on the ground floor, the lift controller should open the door.
- Peter presses the sixth floor button. The press button event is sent to the lift controller. The lift controller will then wait for timeout, close the door, controls the motor to go up and wait for the arrival event. When the lift arrives at the sixth floor, the lift controller will open the door. In the sequence of actions, the lift controller interacts with the door and the motor objects.

Figure 5.49. Sequence diagram for scenario 1



Based on the sequence diagram in Figure 5.49, develop a partial state diagram for the lift controller (Figure 5.50). Figure 5.51 illustrates a partial state diagram for the lift controller from the sequence diagram (Figure 5.49). Develop the state diagram of the lift controller or other control object by adopting the following process:

- Since a state represents a duration in time, treat the time between two consecutive incoming messages to the lift controller as a state.
- If an action takes a significant length of time to complete treat this period of time as a state. For example, the lift will take a significant length of time to go up or down. The time spent going up or down can be considered as a state.
- The controller should be in the state before the scenario takes place.
- The lift controller may transit from one state to another when it receives incoming messages or events. If the transition is not unconditional, we can determine the guard condition for the transition from the conditions of the scenario.

Figure 5.50. A partial state diagram for lift controller

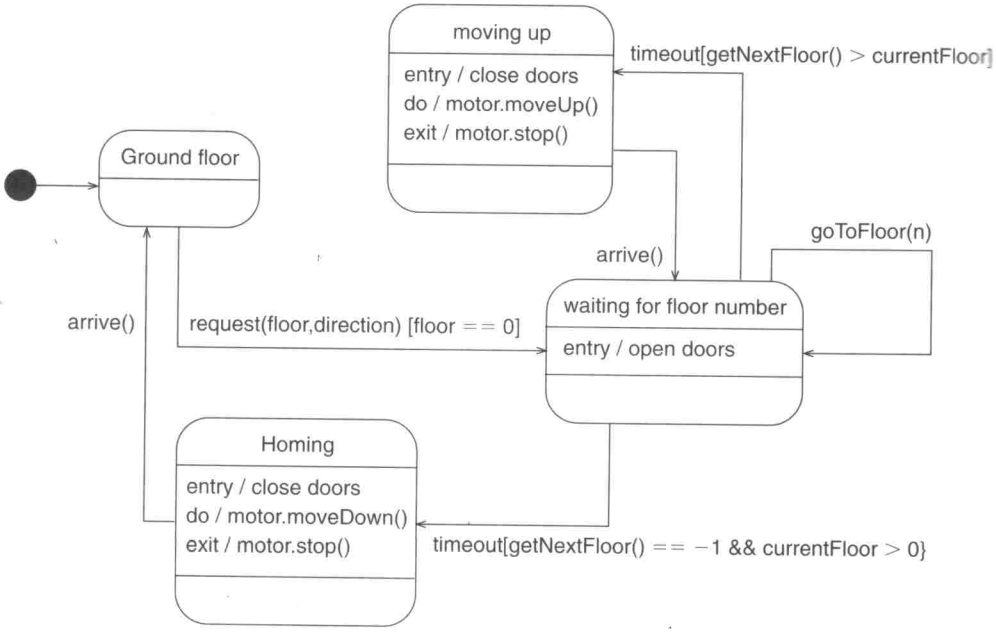
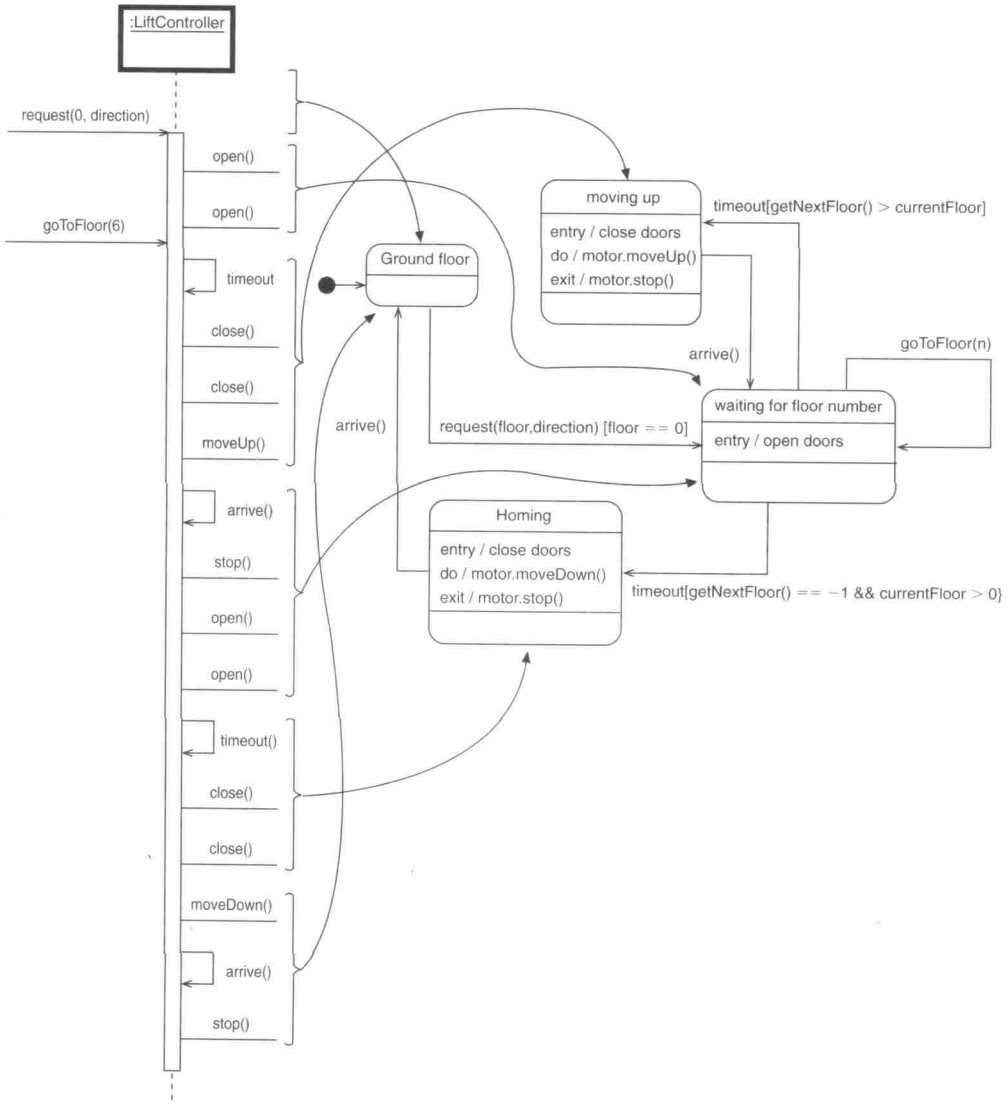


Figure 5.51. Partial state diagram from sequence diagram for lift controller



For example, the state diagram of the lift controller for Scenario 1 can be developed accordingly:

- Before any interaction occurs, the lift is on the ground floor with its doors closed; the lift controller is in the *ground floor* state before receiving any messages or events.
- When the lift controller receives the request message from the UP button on the ground floor, the lift controller opens the door and changes its state to *waiting for floor number*, where it waits for the passenger to press the target floor number. Note that the lift passenger may press the floor number which he/she is currently on.
- The passenger presses the sixth floor button and the lift controller will not do anything until the timeout event is received. Here, the lift controller's state does not change.
- When the lift controller receives the timeout event, the lift controller closes the door, moves the motor to the UP direction because the destination floor is higher then the current floor and then changes its state. Since the action of going up takes a significant length of time, we name this state as *moving up*. The guard condition is that the destination floor number is greater than the current floor.
- When the lift controller receives the arrival event, the lift stops the motor, opens the doors and waits for the passenger to press the destination floor again. Hence, the lift controller changes back to the *waiting for floor number* state.
- When there is no request (no passenger), the lift controller receives the timeout event and the lift controller will move the lift back to the ground floor. We name this state as *homing*.

Scenario 2

A passenger, Mary, is on the sixth floor of the building. She presses the DOWN button at the lift lobby to call for a lift and waits. The lift, which is on the ground floor, then goes up to the sixth floor. The lift stops at the sixth floor and opens. Mary walks into the lift and presses the ground floor button on the control panel in the lift. The lift closes, goes down and stops at the ground floor. The lift opens. It waits for a short while (ten seconds) and then closes the door.

The sequence diagram for Scenario 2 is shown in Figure 5.52. Based on the sequence diagram, develop a partial state diagram for the lift controller as shown in Figure 5.53. Figure 5.54 illustrates how to develop a partial state diagram for the lift controller from the sequence diagram Figure 5.52.

Figure 5.52. Sequence diagram for Scenario 2

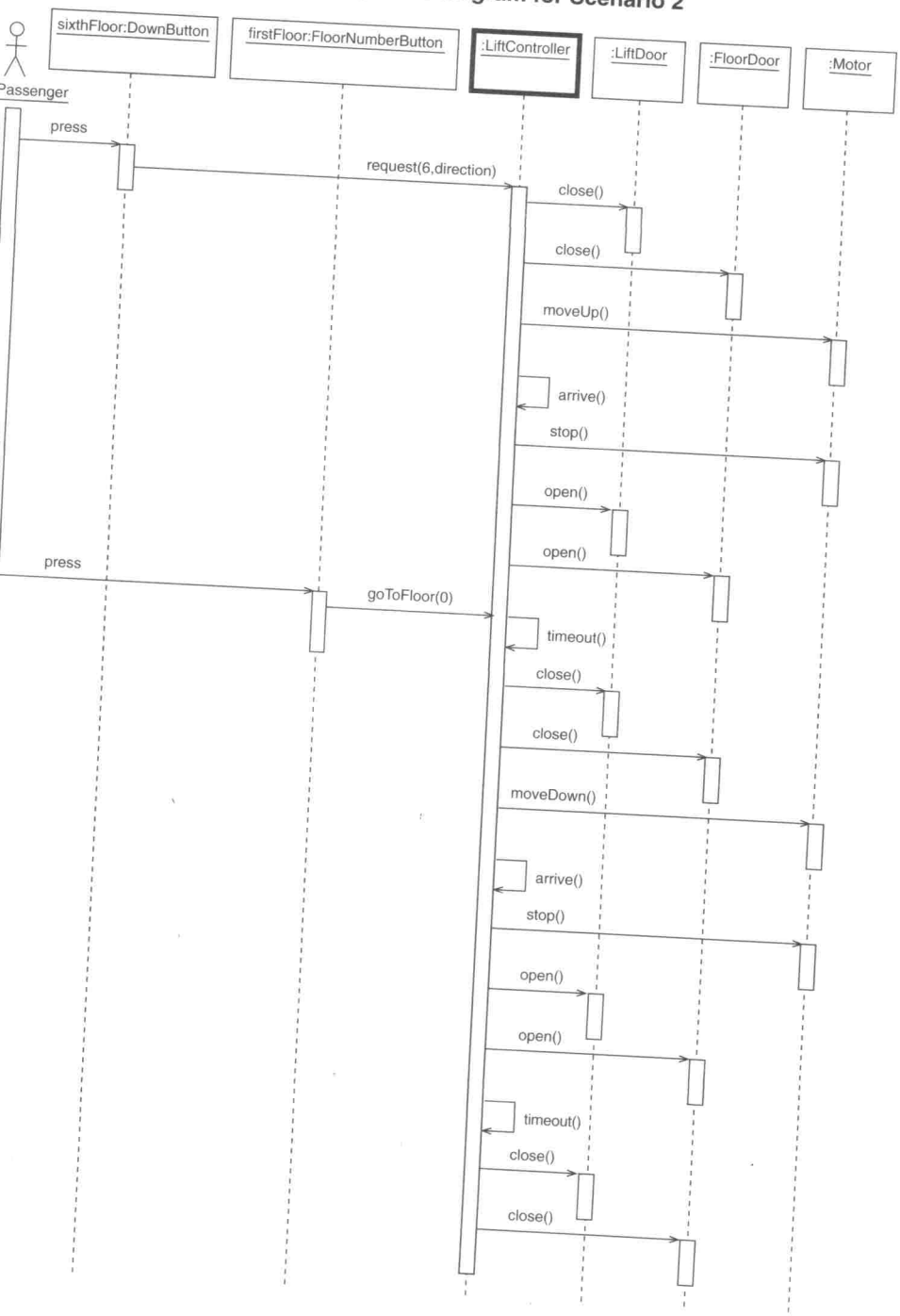


Figure 5.53. Partial state diagram for Scenario 2

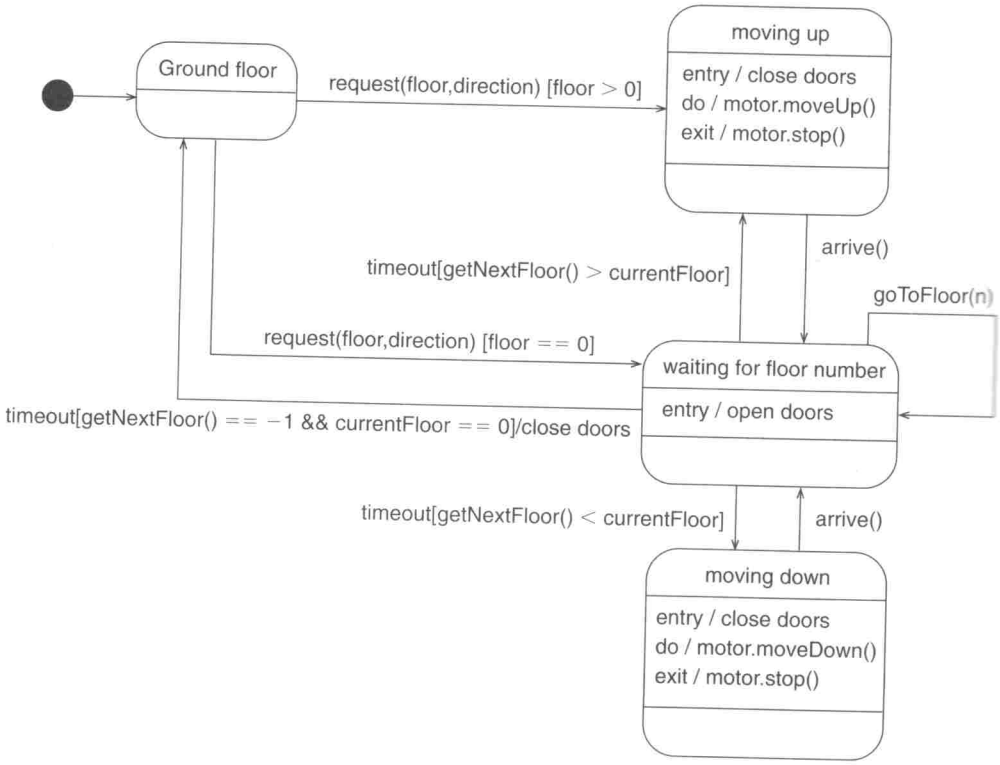
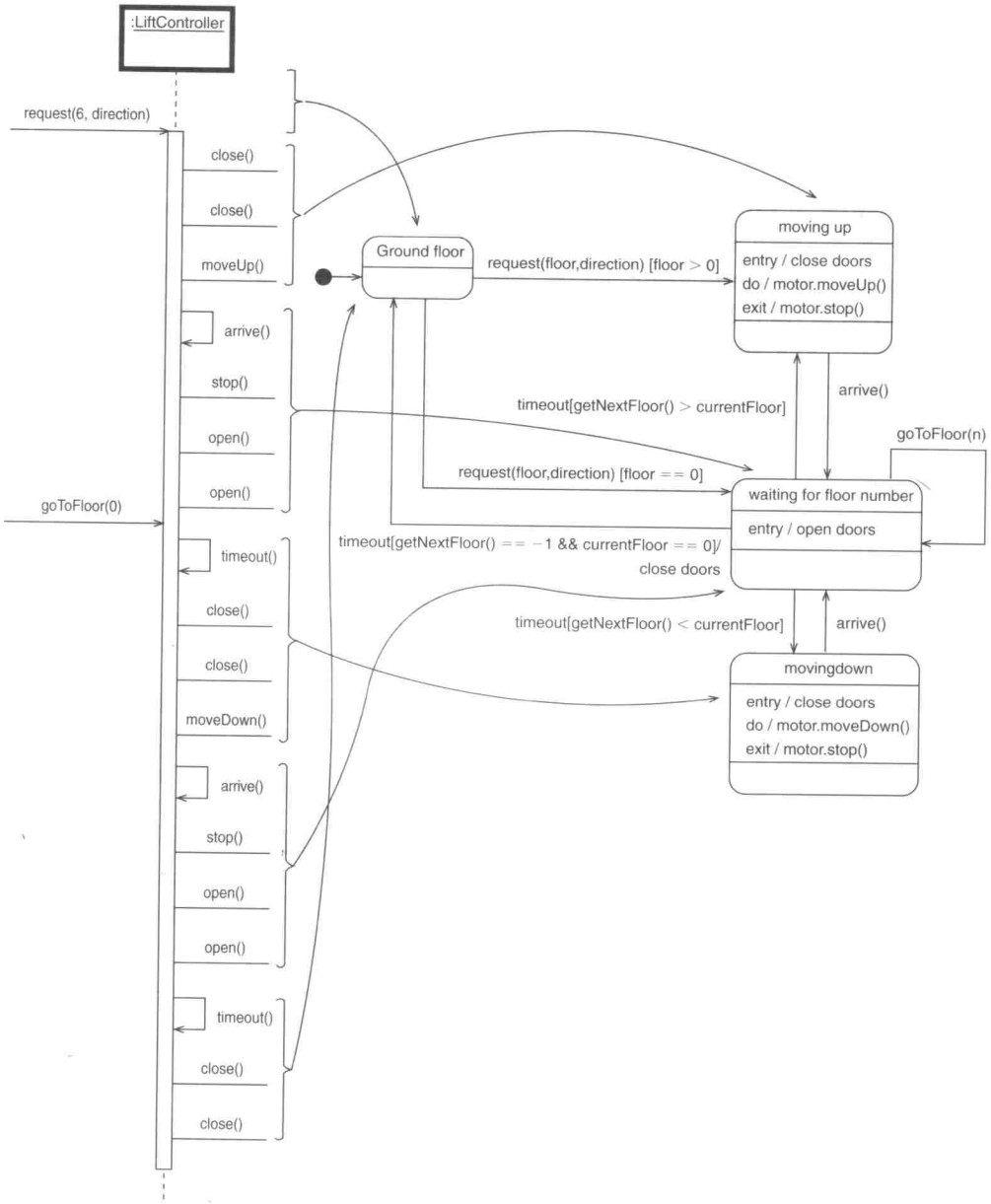


Figure 5.54. Developing a state diagram from sequence diagram for lift controller



It is possible to develop the complete state diagram of the lift controller by combining all the state diagrams of different scenarios via the union of their states and transitions. If there are two transitions with the same source state, target state and event, combine these transitions to become a single transition with a guard condition which is the union of guard conditions of the original transitions. By combining Figures 5.50 and 5.53, we can develop the complete state diagram of the lift controller (see Figure 5.55).

Based on the sequence diagrams for the scenarios and the state diagram of the lift controller, the class diagram is refined to provide more information about the dynamic behavior of the system (see Figure 5.56). Methods are declared in individual classes to implement the messages received by individual classes in the sequence diagrams or state diagrams.

The complete source code of the lift control system in Java can be found in the Appendix.

Figure 5.55. State diagram of lift controller

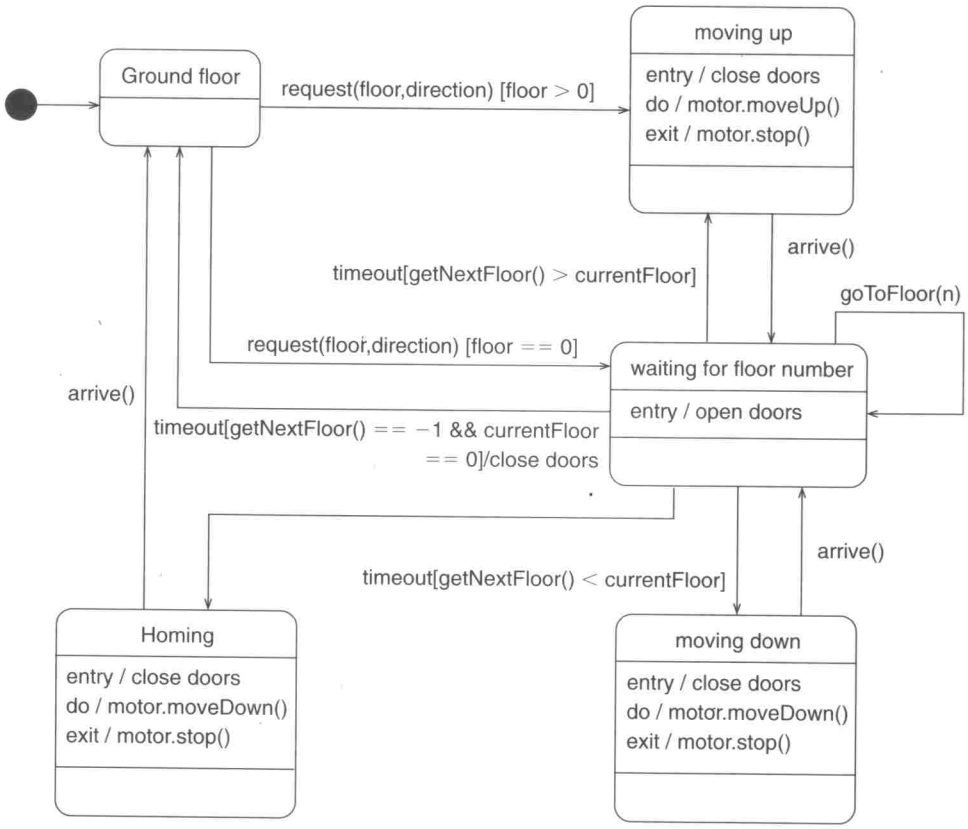
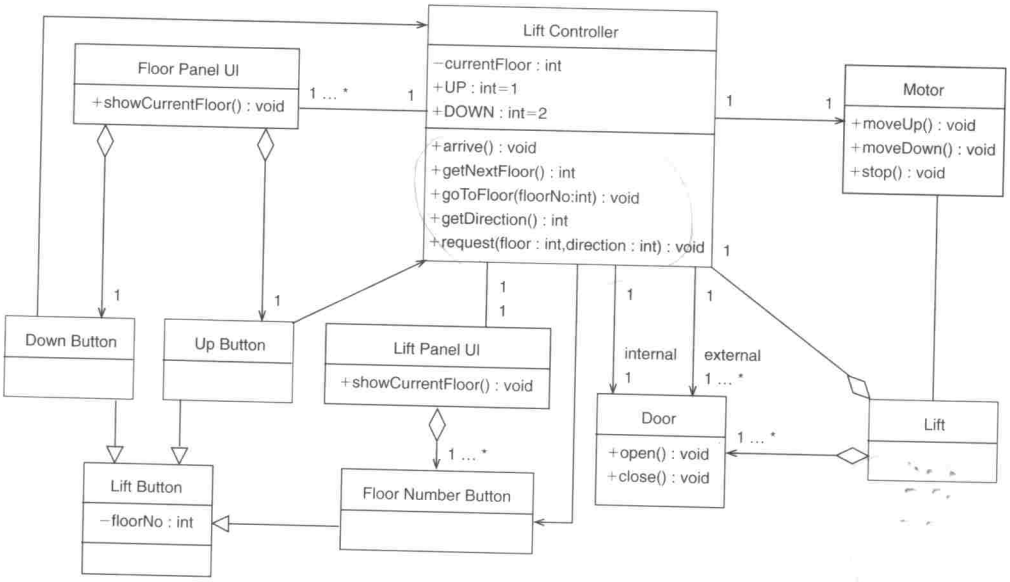


Figure 5.56. Designing class diagram of lift control system



Summary

In this chapter, a number of techniques for the implementation of the more commonly used UML diagrams in Java were presented. We have described how to implement an individual class and the different kinds of associations between classes. Then we illustrated how to implement activity diagrams, state diagrams, sequence diagrams and collaboration diagrams. We have also shown how we can develop the state diagram of a control object from the sequence diagrams using the lift control system case study.

Exercises

Q1. Describe two ways to implement a one-to-many association between two classes. Discuss their pros and cons.

Q2. Using only two Java classes, write a Java program fragment for the implementation of the class diagram below. Include all the methods required for the implementation of the association.