# 7 State Diagrams

## Learning outcomes

The material and exercises in this chapter will enable you to:

- Explain the role and purpose of state diagrams in object-oriented systems development
- Use a state diagram to identify how a class behaves in response to events
- Draw a simple state diagram.

## Key words that you will find in the glossary:

- action
- event
- state
- transition
- activity
- guard
- superstate

## Introduction

So far, we have looked at how to model the organization and structure of data in the system using a class diagram, and at how to model a series of interactions between objects using sequence and collaboration diagrams. In this chapter we examine the system from a different point of view: how a class is affected by the different use cases in the system and how the objects of the class behave in response to events that affect them. The model that illustrates all possible behaviours of a class of objects is called a state diagram. In the chapter we look at the different components of a state diagram, how these are combined, and how the diagrams are used in the development of a system.

State diagrams are an important technique in object-oriented modelling, but they are not widely used in small information systems, such as Wheels. For this reason, most of the examples and exercises in this chapter do not come from the Wheels case study.

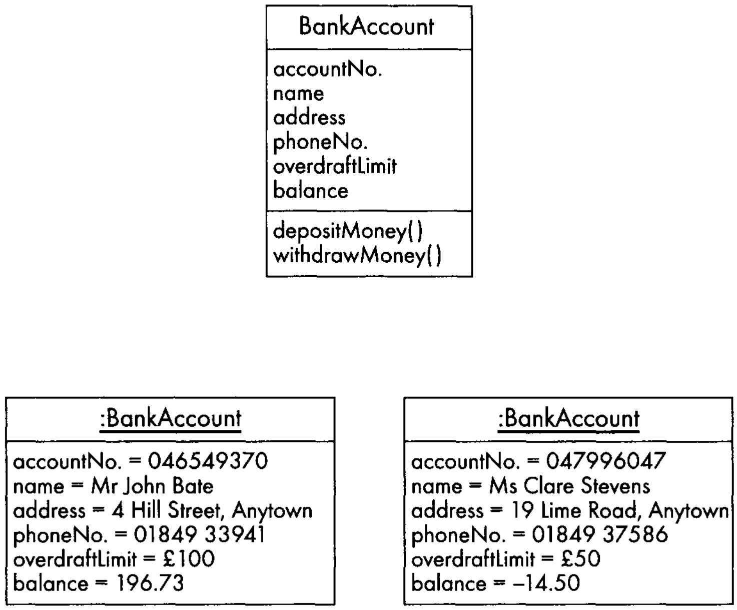

*Figure 7.1* *Class BankAccount and two BankAccount objects*

## States and events

A state diagram models the different states that objects of a class can be in, and the events that cause an object to move from one state to another. In order to be able to draw these diagrams, we therefore need to understand what is meant by state and event in this context.

As we have already seen, a class provides a template or pattern for all the objects of that class. As an example, Figure 7.1 shows a class, BankAccount, and two objects of the class.

Each object of a class such as BankAccount will have the same attributes (although with different values) and the same operations. This means that each object of the class is potentially capable of the same range of behaviours. However, the actual behaviour of an object during the life of the system depends not only on its operations, but also on the events which determine the state that it is in. We can see an example of this if we look at the two BankAccount objects in Figure 7.1. The current balance in John Bate's account is £196.73; it is in the state of being in credit. Clare Stevens' balance, on the other hand, is –£14.50, and her account is therefore in the state of being overdrawn. If John Bate tries to withdraw £40 from his account this will trigger the withdraw money operation and the remaining balance will be £156.73. However, Clare Stevens does not have enough money to withdraw £40 without exceeding her overdraft limit of £50. These two objects are of the same BankAccount class and undergo the same

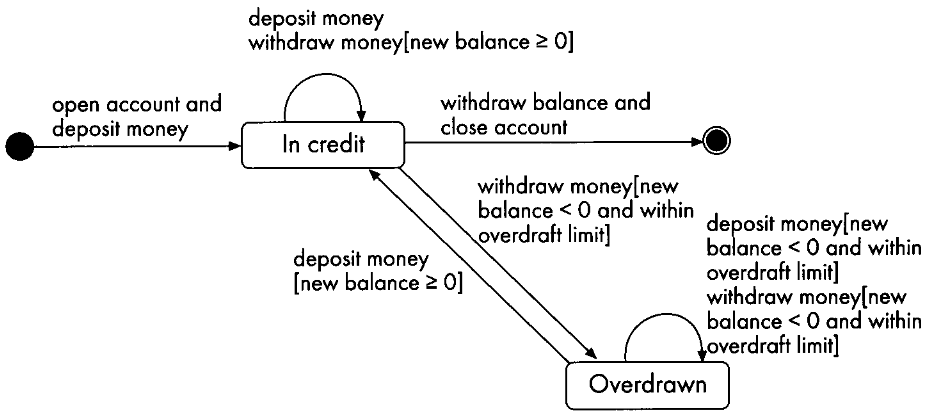
deposit money
withdraw money[new balance ≥ 0]
open account and deposit money
In credit
withdraw balance and close account
withdraw money[new balance < 0 and within overdraft limit]
deposit money [new balance ≥ 0]
deposit money[new balance < 0 and within overdraft limit]
withdraw money[new balance < 0 and within overdraft limit]
Overdrawn

*Figure 7.2 Simple state diagram for the BankAccount class*

event (the attempt to withdraw £40), but they respond differently to the event because they are in different states at the time.

The state of the object here refers to the situation it is in while satisfying some condition (such as a bank account having some money) or waiting for an event (such as someone trying to withdraw or deposit money). An event is something that happens which has significance for the system and affects an object of at least one of the system's classes. We can tell if an object is in a particular state by looking at the values of some of its attributes and its links to other objects. For example, if a BankAccount object is in credit the value of the balance attribute will be a positive amount or zero, but if it is overdrawn the value of balance will be negative. In the Wheels case study, we can tell if a bike is hired out because there will be a link from the Bike object to an active Hire object.

## An example of a simple state diagram

Figure 7.2 shows a simple state diagram for the BankAccount class, illustrating all the different ways in which objects of the class respond to events. The symbols used in state diagrams are illustrated in Figure 7.3

When reading the state diagram in Figure 7.2, we begin from the start state (the filled circle on the left). A state diagram can have only one start state, since all objects of a class are in the same state when created. The event 'open account and deposit money' creates an object of the BankAccount class and causes the object to move from the start state into the 'In credit' state. When a transition between two objects is triggered by an event, the transition is said to fire; it is shown in Figure 7.2 as a labelled arrow between the two states.

While in the 'In credit' state, the object may undergo different types of event, which affect it in different ways; for example as shown in Table 7.1.

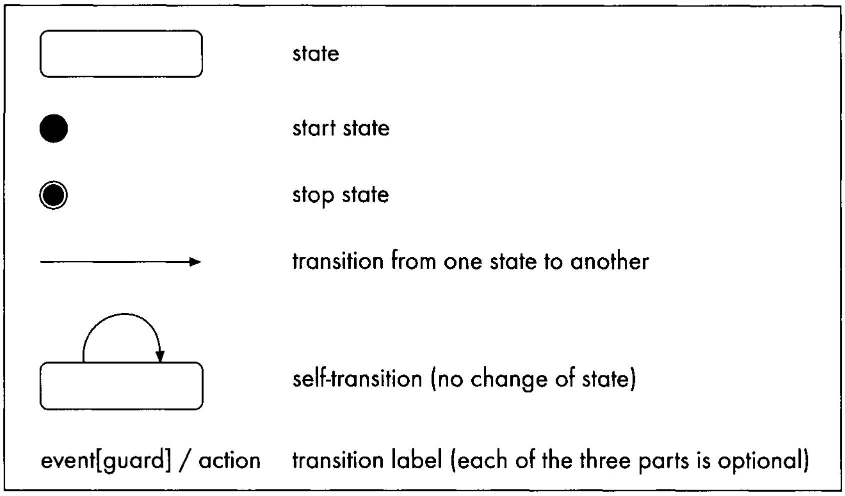
state
start state
stop state
transition from one state to another
self-transition (no change of state)
event[guard] / action
transition label (each of the three parts is optional)

*Figure 7.3* *State diagram symbols*

*Table 7.1:* *Events in the 'In credit' state*

| *Event* | *Result* |
|---|---|
| money is deposited | the account remains in the 'In credit' state |
| all the money is withdrawn and the account is closed | the account moves into the stop state |

While in the 'In credit' state the 'deposit money' event leaves a BankAccount object in the same state. This is known as a self-transition and can be seen as a loop on the 'In credit' state in the diagram in Figure 7.2.

Withdrawing all the money and closing the account causes a BankAccount object to move from the 'In credit' state to the stop state.

Another event which a BankAccount object may undergo while in the 'In credit' state is a 'withdraw money' event. This event can occur with different conditions or guards and therefore may affect the object in different ways (see Table 7.2). It is important to note that the guards relating to an event coming out of a state must be mutually exclusive. This is to ensure that there is no ambiguity about how an object responds to the event. The guards are shown in the state diagram in square brackets.

One of the things that can happen while a BankAccount is in the 'Overdrawn' state is that money may be deposited. This event ('deposit money') can have different conditions or guards and so affect the object in different ways (see Table 7.3).

*Table 7.2: In the 'In credit' state, the event 'withdraw money' can have different results*

| *Event* | *Guard* | *Result* |
|---|---|---|
| money is withdrawn | the new balance is greater than or equal to zero | the account remains in the 'In credit' state |
| money is withdrawn | the new balance is less than zero and within the overdraft limit | the account moves into the 'Overdrawn' state |

*Table 7.3: In the 'Overdrawn' state, the event 'deposit money' can have different results*

| *Event* | *Guard* | *Result* |
|---|---|---|
| deposit money | the new balance is still less than zero | the account remains in the 'Overdrawn' state |
| deposit money | the new balance is zero or more | the account returns to the 'In credit' state |

Another event that can occur in the 'Overdrawn' state is 'withdraw money'. This has a guard '[new balance < 0 and within overdraft limit]', which means that you can still withdraw money while in the 'Overdrawn' state as long as you do not exceed your overdraft limit.

## Constructing a state diagram

In our second example we show you how to build a state diagram.[1] You can find a list of all the steps involved in the summary at the end of the chapter. This example concerns a Human Resources system, where one class, Job Application, is complex enough to justify drawing a state diagram. The diagram will illustrate all the different possible behaviours of objects of the Job Application class.

1. *It is also possible to draw a state diagram starting from the interaction diagrams; for details of how to do this, see Bennett* et al., *(2002).*

A Job Application object is created when an application form is received and the details recorded. The application will then be read by the manager and may be shortlisted or rejected. If rejected, the application is filed for six months. At the end of this time it is discarded. If it is shortlisted, interview details are sent out and the interview is usually confirmed by the applicant. Once the interview has taken place, the applicant may not be successful; in this case a rejection letter is sent and the application is filed for six months and then discarded. If the applicant is offered the job, an offer letter is sent. If the offer is rejected by the applicant the application is filed for six months, and then discarded; if accepted, the application terminates and other procedures take over. The applicant may withdraw at any time during the application process.

In order to draw a state diagram, we need to sort out the events that can occur and the different states that a Job Application object can be in (see Table 7.4). An object always begins life in the start state, before anything happens to it.

We can see from the list that this diagram will be more complex than the previous BankAccount example, as it not only has more states, but there are three different ways in which a stop state may be reached. Multiple stop states are common in state diagrams, as the way an object ends its life will depend on the specific series of events that it undergoes. In contrast, there is only ever one start state on a state diagram, as all objects of a class are created in the same way.

A number of the events that appear separately in the list are actually the same event, but with different conditions, for example the 'read by manager' event has the conditions 'rejected' and 'shortlisted'. These conditions will be represented in the state diagram in square brackets in the guard section of the relevant transition labels.

We should also check at this stage to see if there are any actions that the system has to perform in response to an event. These will be included in the labels on the relevant transitions. In the Job Application example there are two actions, 'send rejection letter' and 'send offer letter'.

We start to construct the state diagram by beginning with the start state, the event that creates a Job Application object, and the state that the object moves into. Figure 7.4 shows the first stage of the diagram.

We can build up the diagram by deciding what events can happen to a Job Application object while it is in the 'Application logged' state and adding them. Figure 7.5 shows the next stage in the process.

Table 7.4: *Events and states for objects of the Job Application class*

| *Event* | *State* |
|---|---|
| | start state |
| application form received and details recorded | Application logged |
| read by manager (rejected) | Filed |
| read by manager (shortlisted) | Shortlisted |
| interview details sent | Shortlisted |
| interview confirmed | Shortlisted |
| interview (unsuccessful) | Filed |
| interview (successful) | Job offered |
| offer rejected | Filed |
| application discarded (after six months) | stop state |
| offer accepted | stop state |
| applicant withdraws | stop state |

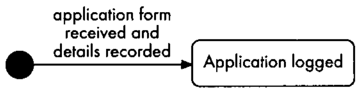

*Figure 7.4* *First stage of the state diagram for the Job Application class*

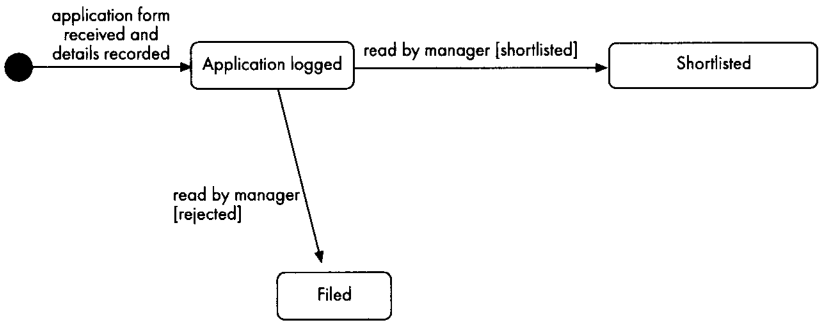
application form received and details recorded
Application logged
read by manager [shortlisted]
Shortlisted
read by manager [rejected]
Filed

*Figure 7.5* *Next stage of the state diagram for the Job Application class*

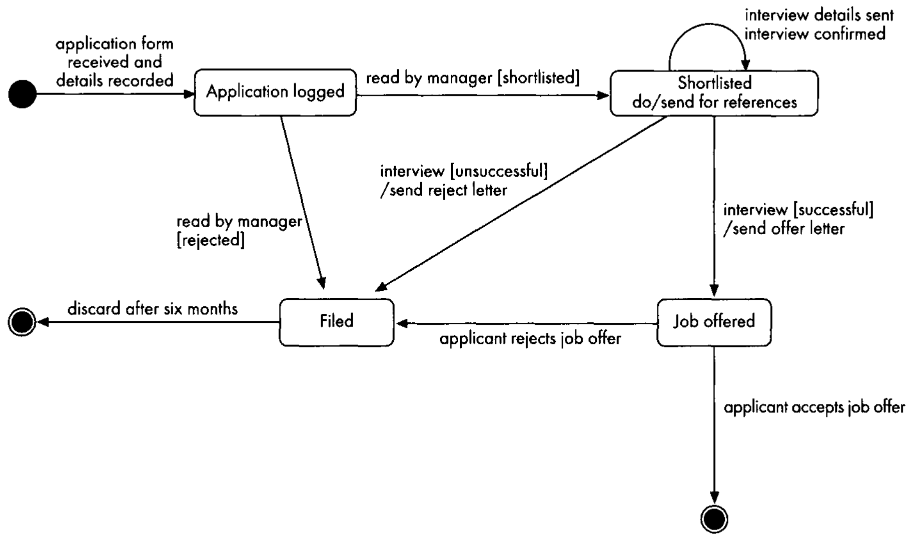
application form received and details recorded
Application logged
read by manager [shortlisted]
Shortlisted do/send for references
interview details sent
interview confirmed
interview [unsuccessful] /send reject letter
read by manager [rejected]
interview [successful] /send offer letter
discard after six months
Filed
Job offered
applicant rejects job offer
applicant accepts job offer

*Figure 7.6* *State diagram for the Job Application class*

We work through the events and states in the list and add them to the diagram, until every item on the list has been included, then we go back to make sure that we have not forgotten any of the guards or actions that should be included in transitions. It is worth noting here that actions can also be contained in states (indicated by the keyword 'do/...' in the state label). This type of action is usually referred to as an activity; it is ongoing (not instantaneous) and can be interrupted by an event. For example, there might be an activity 'send for references' associated with the 'Shortlisted' state.

The state diagram at this stage is shown in Figure 7.6.

There is still one event that we have not included in the diagram. The description of the behaviour of the Job Application class states that an applicant may withdraw at any time. In order to include this in the diagram in Figure 7.6, we would need to add a third stop state and draw transitions to it with the event 'applicant withdraws' from each of the four states on the diagram. This would make the existing diagram cluttered and very difficult to read. In order to avoid clutter, we can draw a superstate round the main body of the diagram, and show a single 'applicant withdraws' transition from it, indicating that an applicant can withdraw at any time during the application process.

Finally, we need to check the completed diagram against the original description of the behaviour of the Job Application class, in order to confirm that it is an accurate representation. The completed diagram with the superstate is shown in Figure 7.7.

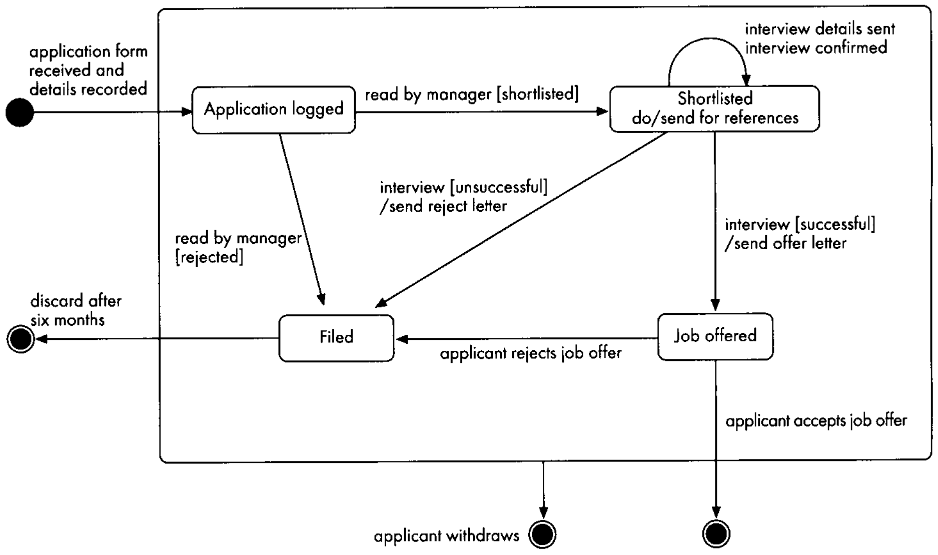
application form received and details recorded
Application logged
read by manager [shortlisted]
Shortlisted do/send for references
interview details sent
interview confirmed
interview [unsuccessful] /send reject letter
read by manager [rejected]
interview [successful] /send offer letter
discard after six months
Filed
applicant rejects job offer
Job offered
applicant accepts job offer
applicant withdraws

*Figure 7.7* *Completed state diagram with superstate for the Job Application class*

## An example from the Wheels case study

For a final example of a state diagram, we return to the Wheels case study. Most of the classes in the Wheels system are relatively simple; the only one where the behaviour is complex enough to merit a state diagram is the Bike class.

As with the previous example, we begin by identifying the events in the system that can affect an object of the Bike class and the different states that an object can be in. The information that we use here to identify the events and states comes from Chapter 2, Requirements for the Wheels case study system.

There may also be some actions that we will need to consider, but we will first construct the basic state diagram. As before, we begin with the start state, the event that creates a Bike object, and the state that the object moves into. Figure 7.8 shows the initial stage of the diagram.

We now build up the diagram as in the previous example, working through the events and states in the list and adding them to the diagram, until every item on the list has been included (see Table 7.5). Some of the events may occur more than once, when the object is in different states; for example, a bike may be sold when it is new or available for hire, but not when it is on hire or under repair. The 'sold' event will appear in the diagram as a transition to a stop state from both the 'New bike' state and the 'Available for hire' state.

We should also take particular note of any events which appear as separate on the list, but which should be represented in the diagram as the same event with different guards. In this case the events 'minor

*Table 7.5: Events and states for objects of the Bike class*

| *Event* | *State* |
|---|---|
| | start state |
| bike purchased | New bike |
| bike number is assigned | Available for hire |
| customer hires bike | On hire |
| customer returns bike | Available for hire |
| minor damage to bike | Under repair |
| major damage to bike | stop state |
| bike repaired | Available for hire |
| bike lost or stolen | stop state |
| bike sold | stop state |
| bike scrapped | stop state |

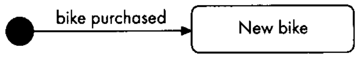

*Figure 7.8 First stage of the state diagram for the Bike class*

damage to bike' and 'major damage to bike' will be represented as one event, 'bike damaged', with guards [reparable] (leading to the 'Under repair' state) and [irreparable] (leading to a stop state).

Figure 7.9 shows the intermediate diagram representing events and states, but without any actions.

The next stage is to check whether we need to include a superstate to cater for events that can occur at any stage in the life of an object. In this example 'bike lost or stolen' is such an event, so we represent this event by a transition from a superstate to a stop state.

Finally, we need to consider whether any actions should be included on the state transitions. For this we will need to look at Chapter 2 again. We discover that the events 'bike damaged/[reparable]' and 'bike damaged/[irreparable]' both have an action 'extra charge to customer', so this action should be added to the relevant transition labels on the diagram.

The completed state diagram for the Bike class is shown in Figure 7.10. This should now be validated against all the information that we have gathered about the behaviour of objects of the Bike class.

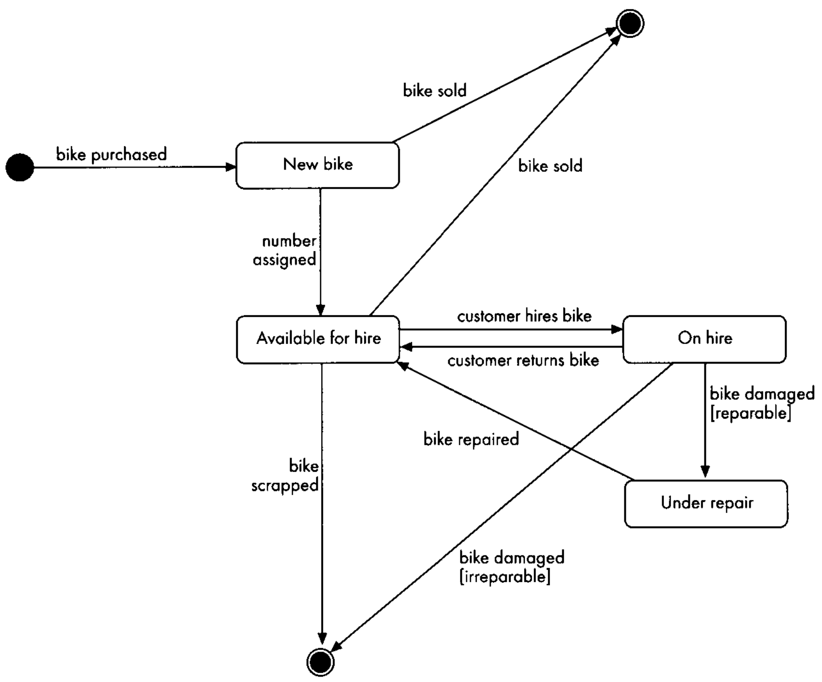

*Figure 7.9 Intermediate stage of the state diagram for the Bike class*

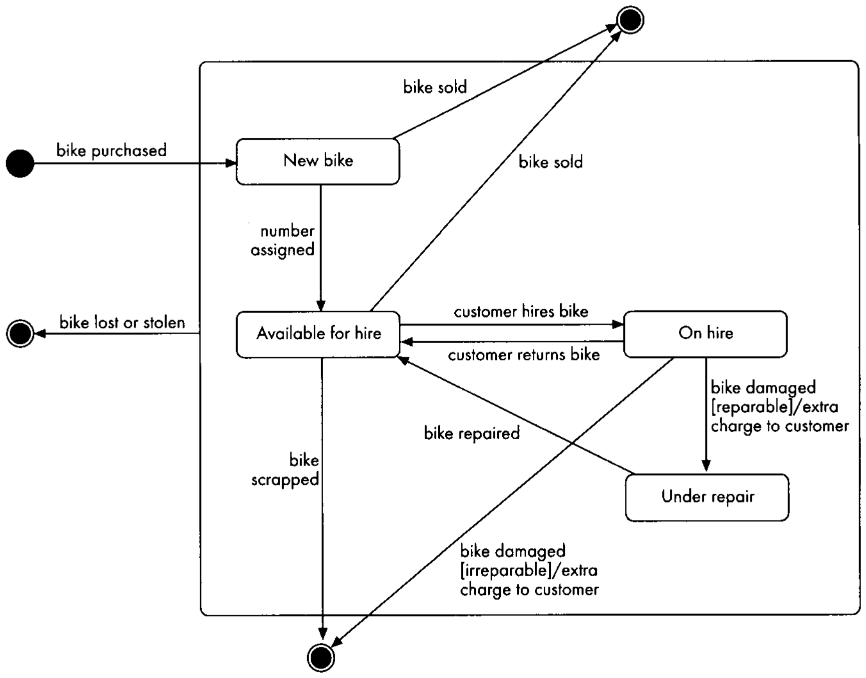

*Figure 7.10 Completed state diagram for the Bike class*

## Using state diagrams in system development

State diagrams model the system from the point of view of a single class and the events that can affect the objects of the class. They show all possible behaviours of objects of a class, and record the ordering of events, for example in the Wheels system a bike must be assigned a number before it can be hired. This information about timing constraints is vital for our understanding of the system, and is not recorded in any of the other system models that we cover in this book.

Although state diagrams are a very useful and important modelling technique, it is not necessary to draw one for every class. In most systems, complexity arises from interaction between objects of different classes, as modelled in sequence and collaboration diagrams (see Chapter 6). It may well be that in any system, particularly an information system, only a few classes will display dynamic behaviour and so need a state diagram to model what happens. This is the case with a system such as Wheels, where most classes have objects that undergo only a restricted set of events and all the objects respond to the same event in the same way. For example, all Customer objects in the Wheels case study system respond to events that happen to them in the same way, although with different values: recording details, finding an associated Hire object, displaying customer details and amending customer details. Their response to events does not depend on what state they are in. For this sort of class which has relatively simple behaviour there is not a lot to show on a state diagram. However, other types of computer system, for example process control or communication systems, frequently have a number of classes whose dynamic behaviour is extremely complex. For this sort of class it is important to document all the possible behaviours of the objects of the class by means of a state diagram.

As with all models that are produced as part of the development process, it is important to check that state diagrams are consistent with other diagrams. A state diagram of a particular class should be checked against interaction diagrams which involve objects of the class to ensure that all the events in the state diagram appear in the interaction diagram as an incoming message to the object. It should also be checked against the class diagram to make certain that every event and action corresponds to an operation on the relevant class.

## Technical points

*Different types of event.* Not all events occur because of outside influences. For example, an event can happen in the course of time. In the state diagram, this is represented by the keyword 'after', for

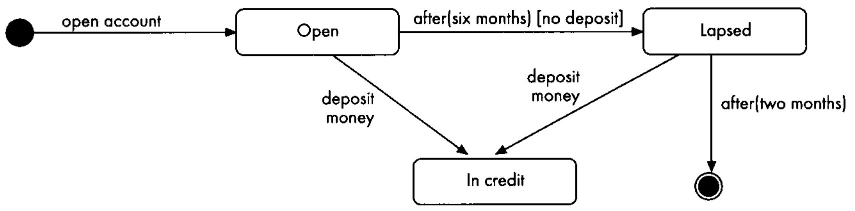

*Figure 7.11* *Modified BankAccount example showing use of keyword 'after'*

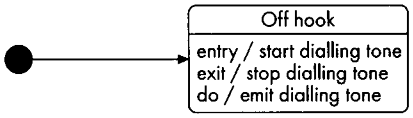

*Figure 7.12* *Off hook state showing exit and entry events and activity*

example 'after (six months)'. In the bank account example we might find, on further investigation, that an account can be opened without a deposit. It is then in the 'Open' state. If a deposit is made it goes into the 'In credit' state. If no deposit is made after six months, it goes into a 'Lapsed' state. It will be deleted two months later unless a deposit is made. This is illustrated in Figure 7.11.

An event can also occur when a certain condition is satisfied; this is represented by the keyword 'when', as for example 'when (all items in stock)'.

Sometimes events with actions occur every time a state is entered or exited. For example, every time a phone enters the state where it is off the hook, but a number has not yet been dialled, it starts emitting a dialling tone which ends when the first number is dialled. These are known as entry and exit events and are shown in the label for the state with the associated action, as for example 'entry / start dialling tone' (see Figure 7.12). Behaviour that lasts for the duration of the state is called an activity and is modelled using the keyword 'do'. Unlike actions associated with events, activities can be interrupted.

*Nested states.* If a state diagram becomes too complex, it can be simplified by nesting related sets of substates. For example, the 'New bike' state in Figure 7.10 has three substates: 'Bike check', 'Assign Wheels number', 'Register bike on computer'. These are modelled as nested substates in Figure 7.13. If we showed the detail of the substates on the main diagram it would become too cluttered, but it is sometimes useful to be able to decompose states

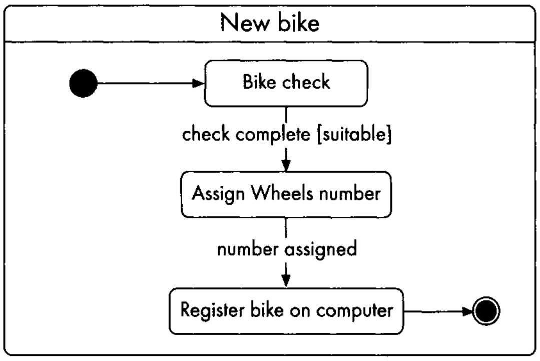

*Figure 7.13* *New bike state showing internal nested substates*

to study the internal detail. Notice that the events and states within 'New bike' have their own start and stop states. The nested state diagram can be referenced in the higher level state using the keyword 'include'. For more details about nested substates, see Bennett *et al.*, (2002), Chapter 11.

*Concurrent state diagrams.* Sometimes the behaviour of an object depends on two independent sets of substates. For example, in a more complex version of the original Job Application state diagram (see Figure 7.6), there might be a set of substates dealing with setting up an interview, and a parallel set of substates to do with obtaining references. This can be shown by drawing a concurrent state diagram – see Figure 7.14. For more details about concurrent states see Fowler (2000), Chapter 8, and Bennett *et al.*, (2002), Chapter 11. It is important to remember, however, that too much detail in one diagram can make the diagram cluttered and difficult to read; it is often simpler and more effective to draw separate diagrams.

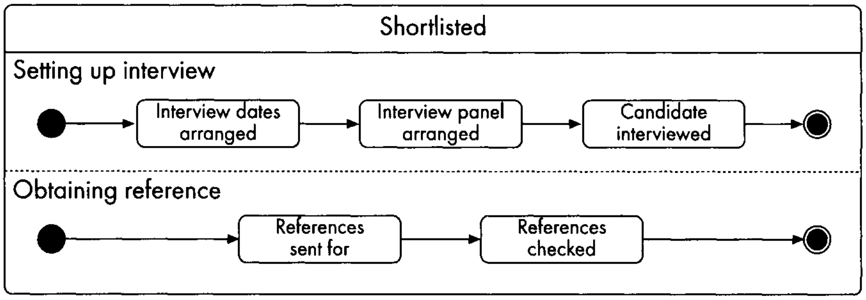

*Figure 7.14* *Concurrent state diagram for the 'Shortlisted' state in the Job Application example*

## Common problems

1 Can I draw a state diagram for the whole system?

No – a state diagram normally only models the behaviour of the objects of a single class. One of the most common mistakes that students make when learning the technique is to try to model the behaviour of the whole system in a single state diagram.

2 How do I know if I need to draw a state diagram for a particular class?

In order to decide whether or not to draw a state diagram, you need to look at how objects of the class behave in response to events; you can see this by studying the lifelines of the objects in all the relevant interaction diagrams, i.e. all the ones in which objects of this class feature. When you look at the lifeline of an object in a sequence diagram, you can see all the events that happen to it (i.e. the messages sent to it) and how many there are; you can also see whether or not the object always responds in the same way. This is even more apparent in collaboration diagrams as the way they are drawn emphasizes all the messages coming to an object. In fact examining interaction diagrams is one of the main starting points when drawing state diagrams. A good (i.e. representative) set of interaction diagrams will show all the events that can happen to an object during its lifetime (all the messages that can be sent to it) and all the different ways it can respond. From this a list of events can be drawn up like the ones we compiled for the Job Application and Bike state diagrams. State diagrams and interaction diagrams look at the same events but from a different viewpoint. An interaction diagram shows how the execution of a particular scenario affects all of the objects involved. A state diagram looks at a particular class of objects and shows how all of the scenarios affect them. So when your state diagram is complete you should be able to take each scenario in turn and trace through the state diagram following the sequence of events that affect that class.

Students often try to draw a state diagram for a class that is not complex enough to need one, such as the Customer class in the Wheels system. It is only useful to draw a state diagram in cases where the way an object of that class responds to an event depends on the state it is in. This is what textbooks mean when they refer to an object having dynamic behaviour. Only classes with dynamic behaviour are worth modelling with a state diagram.

3 How can I tell the difference between states and events?

Students often get confused about what is a state and what is an event, and it is sometimes difficult to make the distinction between them. The main difference is that an event is regarded as being almost instantaneous and uninterruptible, whereas a state lasts longer – it has duration. For example, in the Wheels system 'bike damaged' is regarded as an event that cannot be interrupted because it is in the past and has happened. On the other hand, 'Under repair' is a continuous state that a bike may be in for some time. It is helpful to label events and states with completely different names; for example, in Wheels we could have an event 'bike hired' leading to a state 'Hired', but it is much clearer to label the event 'customer hires bike' and the state 'On hire'.

4 What is the relationship between state diagrams and interaction diagrams?

The two types of diagram show related information (the behaviour of objects in use cases), but the emphasis is completely different. A state diagram shows how the different objects of a single class behave through all the use cases in which the class is involved. An interaction (sequence or collaboration) diagram concentrates on a single use case and shows how all the objects involved behave during the use case (see also the answer to Question 2, above).

5 How do I know whether to include all the fancy stuff, like concurrent states and different types of event?

There is no hard and fast rule for this, but you should remember that, as with all models, there is a risk of including too much detail and making the diagram so cluttered that it is unreadable. If the extra information is important to the overall understanding of how the system works, you should include it, otherwise leave it out. You may also find that the same information can be shown (possibly more effectively) in one of the other types of diagram and in that case it should not be duplicated.

## Chapter summary

This chapter introduces state diagrams, which are used to model the ways in which the objects of a class respond to events that affect them. It describes when to use a state diagram, explains the notation used, and provides guidelines on how the diagrams are constructed. The basic steps that we describe for drawing a state diagram in this chapter are:

- Identify the events that affect an object of the class
- Identify the different states that objects of the class can be in, including the start state and (possibly) multiple stop states
- Check whether any events that are listed separately should be represented as the same event with different conditions (guards)
- Check whether there are any actions that the system must perform in response to an event or whilst in a given state; these should be represented as actions in the transition or state labels
- Begin to construct the diagram from the start state, the event that creates an object of the class, and the state that the object moves into
- Build up the diagram, working through the events and states on the list and adding them to the diagram
- Check that all guards and actions have been included on the relevant transition labels
- Check whether a superstate should be included to cater for events that may occur at any time during the life of an object
- Check the completed diagram against the information that has been gathered about the behaviour of the class.

## Bibliography

Bennett, S., McRobb, S. and Farmer, R. (2002) *Object-Oriented Systems Analysis and Design Using UML* (2nd edition), McGraw-Hill, London.

Britton, C. and Doake, J. (2002) *Software System Development: A Gentle Introduction* (3rd edition), McGraw-Hill, London.

Fowler, M. (2000) *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (2nd edition), Addison-Wesley, Reading, MA.

## Quick check questions

You can find the answers to these in the chapter.

a What aspect of a system is modelled by a state diagram?

b What is meant by 'state' in this context?

c What is meant by 'event' in this context?

d What are the three parts of a transition label? Which parts have to be present?

e What is a self-transition on a state?

f Why must the guards relating to the same event coming out of a state be mutually exclusive?

g When do you need to include a superstate on a state diagram?

h For what types of system are state diagrams generally most useful?

## Exercises

7.1 Burglar alarm.

a When new, a burglar alarm is in a Resting state, and while it is in this state, the alarm may be set. This event moves the alarm into a Set state. While in the set state, the alarm may be turned off, and so returns to the Resting state. Draw a state diagram for the Burglar Alarm class.

b While in the Set state, the alarm may be triggered; this moves it into the Ringing state. From here the alarm may be turned off, and so return to the Resting state. Amend the state diagram you drew in (a) to include this information.

c The alarm may break at any time. Include this information on the diagram using a superstate.

7.2 Estate agent's property.

Figure 7.15 is a state diagram of a property in an estate agent's system. Study the diagram and then briefly describe in clear English what can happen to a property during its life in the system.

NB The superstate in this diagram is slightly different from those in the diagrams in the chapter in that it does not apply to all the states. You should be able to see from the diagram when the vendor can take the property off the market and when this is no longer possible.

7.3 Simple microwave oven.

When new, a microwave oven is initially off. From this state the cooking time may be set and the oven turned on. While the microwave is on, the time can be changed. When the time is up, the microwave turns itself off and gives three

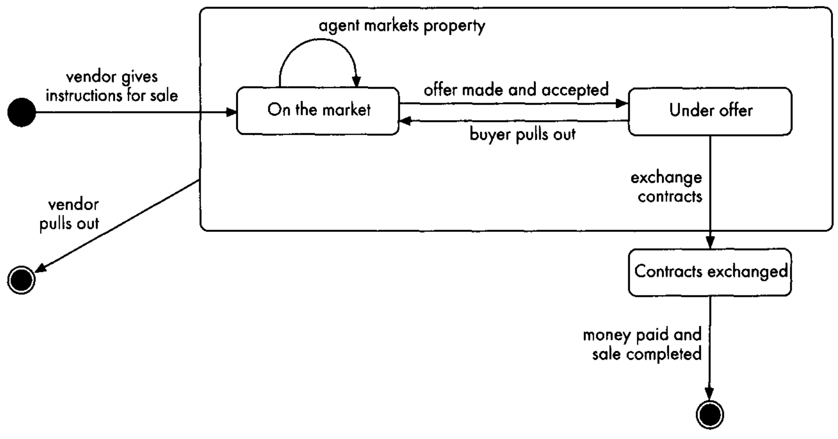

*Figure 7.15* *State diagram for an estate agent's property*

short beeps. Draw a state diagram to represent the behaviour of the simple microwave oven.

7.4 Newsagent's customer.

A newsagent has customers who place regular orders for papers to be delivered and who are billed monthly. If the customer does not pay the bill within four weeks, the newsagent sends a reminder. If the bill is still unpaid after a further two weeks, the newsagent stops deliveries to the customer. Customers can change or cancel their orders, but only if they have paid all bills to date. Draw a state diagram to represent the behaviour of a customer in the newsagent's system.

7.5 Tea and coffee machine.

A tea and coffee machine in an office is initially idle, until a user inserts 50p. At this point the user can press the tea button to select tea, which the machine then dispenses, or the user can insert a further 20p. When 70p has been inserted, the user can press the coffee button to select coffee, which the machine dispenses, before returning to the idle state. The machine may break at any time. Draw a state diagram to represent the behaviour of the tea and coffee machine.

7.6 Internet book order.

Figure 7.16 is a state diagram of an order in an Internet book shop system. Study the diagram and then answer the questions below.

a How does the event 'user selects book' affect the state of an Internet Book Order object?

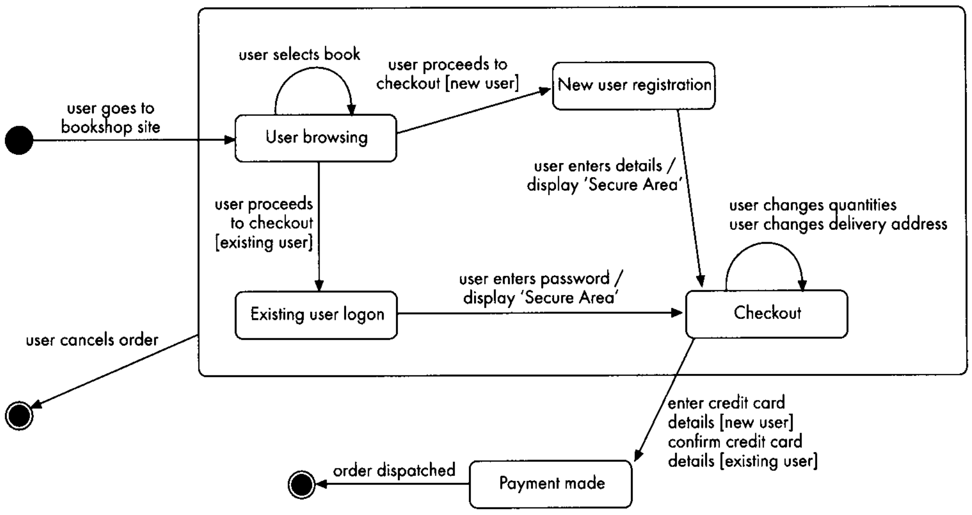
user goes to bookshop site
user selects book
User browsing
user proceeds to checkout [new user]
New user registration
user enters details / display 'Secure Area'
user proceeds to checkout [existing user]
Existing user logon
user enters password / display 'Secure Area'
user changes quantities
user changes delivery address
Checkout
user cancels order
enter credit card details [new user] confirm credit card details [existing user]
order dispatched
Payment made

*Figure 7.16* *State diagram for an internet book order*

b What personal information does an existing user have to enter first?

c At what point is the 'Secure Area' message displayed to a new user?

d If the user wants to send a book to someone else, where can they arrange this?

e Can a user change their mind and cancel once payment has been made?

f In what ways can the life of an Internet Book Order object end?