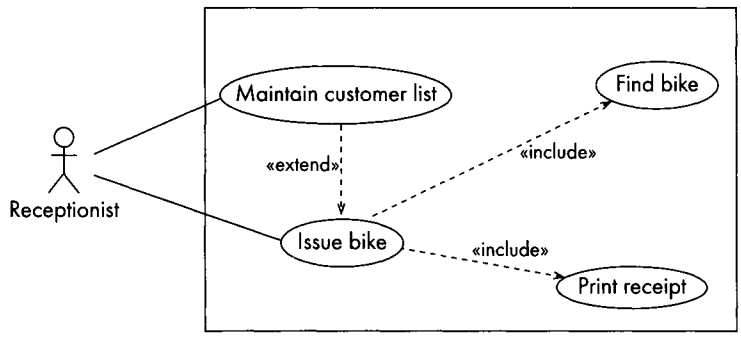

Figure 3.10 *Extract from the Wheels use case diagram showing the «extend» relationship between 'Issue bike' and 'Maintain customer list'*

«include» because we will not always add a new customer, or edit customer details as part of issuing a bike.

Notice that the minor use case is always modelled as an extension of the major one, i.e. the core functionality is specified in the base use case and the additional or exceptional behaviour in the extending use case.

The user sometimes has to select one of several options at a particular point in the use case; in such cases it may be useful to model each option as a separate extending use case. The 'alternative courses' section of the extended use case description (see Figure 3.7) is also used for specifying differences in use case behaviour. Which method you choose is a matter of judgement.

We have drawn a separate diagram (Figure 3.10) to illustrate the refined version of 'Issue bike' as our main diagram is getting a bit cluttered. This effectively makes the point that «include» and «extend» should be used sparingly. It is easy to get carried away with enthusiasm when identifying legitimate uses for these relationships, but the resulting model is not always an improvement.

Documenting «include» and «extend» in the use case description. If we have added «include» or «extend» relationships to a use case diagram, we must document them in the use case description. This is done using the keyword 'initiate'. For example, if the use case 'Issue bike' were to be modelled as specified in Figures 3.9 and 3.10, we would adjust its use case description to document the «include» and «extend» relationships, as in Figure 3.11.

Boundary

In general the concept of the system boundary is important as it delineates the domain of study. The system boundary may be included on a use case diagram (see Figures 3.2 and 3.9). It is shown