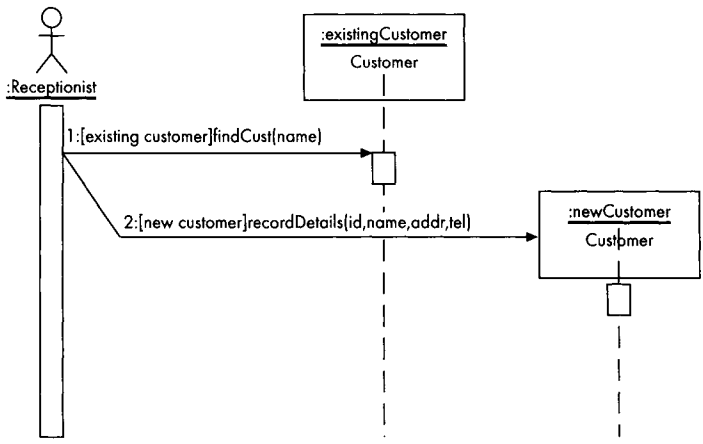

Figure 10.9 *Fragment of diagram showing branching*

Concurrency. Sequence diagrams can also be used to model synchronous and asynchronous messages. All of the messages we have met so far have been synchronous. When an object sends a *synchronous* message, it must wait for a response from the object it calls. The sending object, therefore, cannot continue with its own processing until it gets a response. This is often because it is waiting for data. Up till now, at any given time, we have only modelled situations where a single object is processing. This is not particularly surprising as many information systems are *single-threaded* and designed to run on single processor computers. However, in a multi-threaded system, i.e. one designed to have several processes executing in parallel, it is useful to be able to model asynchronous messages. When an object sends an *asynchronous* message it can invoke an operation on another object without interrupting its own processing; the two objects can process in parallel. This is also known as *concurrent* processing. In Figure 10.10 we model the home-coming of a teenager, chris, after a hard day at school. He is very hungry and makes a bee-line for the fridge. Without pausing in his stride, or rather his hunt for food, he tells his little brother to put the television on so that he can watch the football, asks his baby sister to give him a smile and drops his games kit on the ground telling his mum that he needs it washed by tomorrow. All of these are asynchronous messages, he starts three processes in motion without interrupting his own job which is the search for food. An asynchronous message is indicated by a half arrowhead.

Notice another new notation in the diagram in Figure 10.10; there is a {by tomorrow} constraint on the message washKit(kit). A *constraint* is simply an assertion about the modelling element it