Chapter 15

Modelling States

After taking a break to consider Inheritance and System Architecture, we are now going to return to the design stage of the construction phase and consider state modelling.

State Diagrams allow us to model the possible states that an object can be in. The model allows us to capture the significant events that can act on the object, and also the effect of the events.

These models have many applications, but perhaps the strongest application is to ensure that odd, illegal events cannot happen in our system.

The example given in the introductory chapter of the book (page 31) talks about a situation that seems to happen an awful lot, if local newspapers are anything to go by - a gas bill is sent to a customer who died five years ago!

Carefully written state diagrams should prevent these kind of erroneous events occurring.

Example Statechart

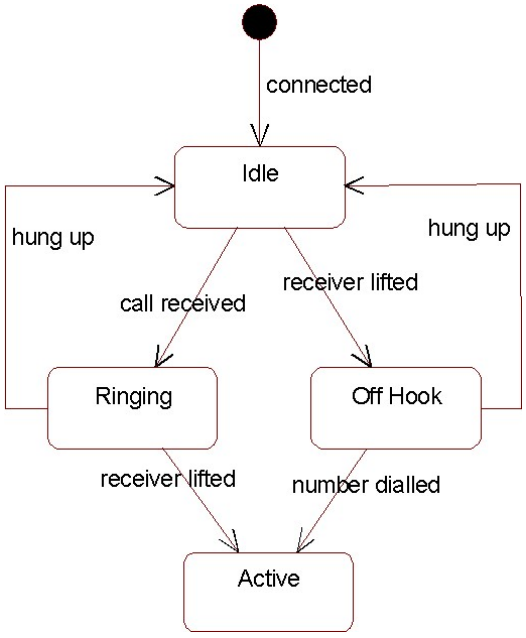

Figure 94 - Example Statechart; Telephone

We will look at the syntax of this diagram in detail shortly, but the basics of the diagram should be obvious. The sequence of events that can occur to the telephone are shown, and the states that the telephone can be in are also shown.

For example, from being idle, the telephone can either go to being "Off the Hook" (if the receiver is lifted), or the telephone can go to "Ringing" (if a call is received).

State Diagram Syntax

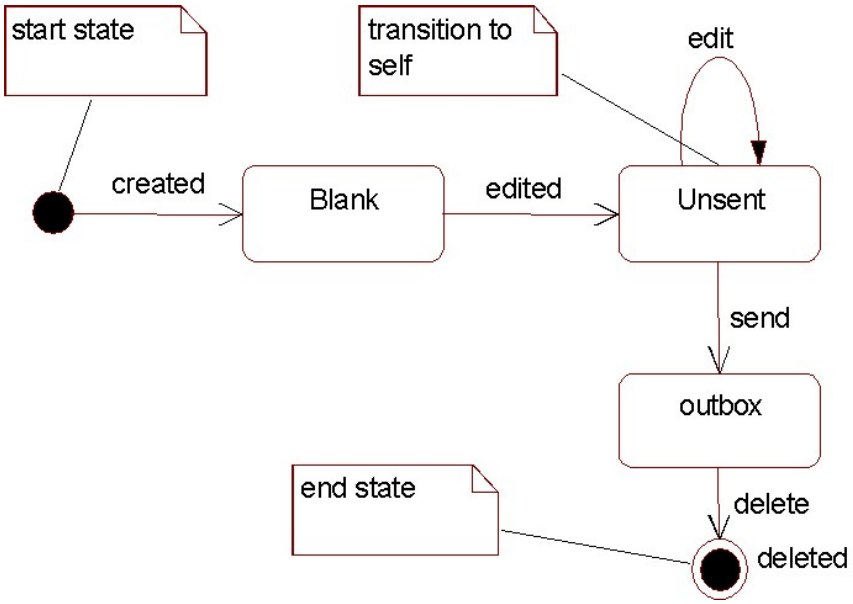

Figure 95 - Syntax of the State Diagram - an E-Mail example

The diagram above shows most of the state diagram syntax. The object will have a start state (the filled circle), describing the state of the object at the point of creation. Most objects have an end state (the "bullseye"), describing the event that happens to destory the object.

Some events cause a state transition that causes the object to remain in the same state. In the example above, the e-mail can receive an "edit" event only if the status of the object is "unsent". But the event does not cause a state change. This is a useful syntax to illustrate that the "edit" event can not happen in any of the other states.

Substates

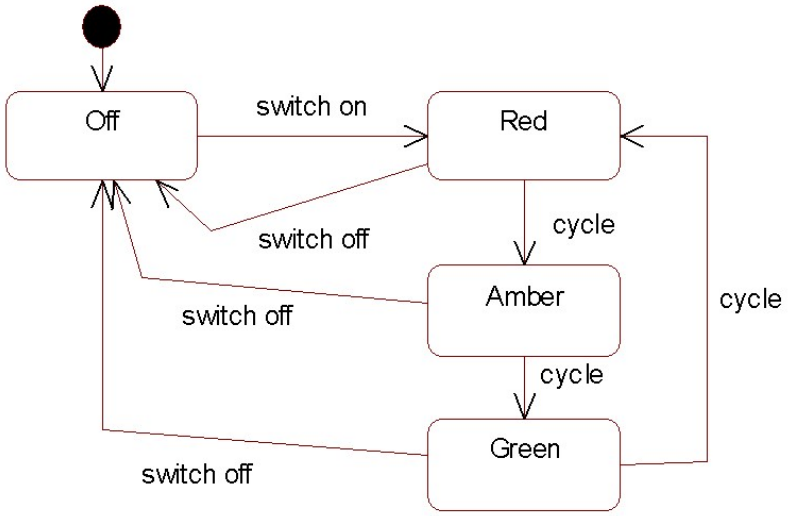

Figure 96 - Messy State Model

Sometimes, we require a model that describes states within states. The above statechart is perfectly valid (describing a traffic light object's states), but it is hardly elegant. Essentially, it can be switched off at any time, and it is this set of events that is causing the mess.

There is a "superstate" present in this model. The traffic light can be either "On" or "Off". When it is in the "On" state, it can be in a series of substates of "Red", "Amber" or "Green". The UML provides for this by allowing "nesting" of states:

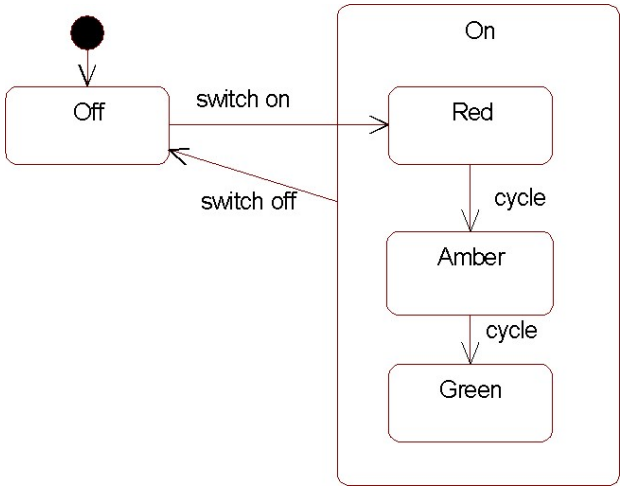

Figure 97 - Simpler state model using substates

Note that in the diagram above, the small arrow pointing in to the "red" state indicates that this is the default state - on commencement of the "on" state, the light will be set to "Red".

Entry/Exit Events

Sometimes it is useful to capture any actions that need to take place when a state transition takes place. The following notation allows for this:

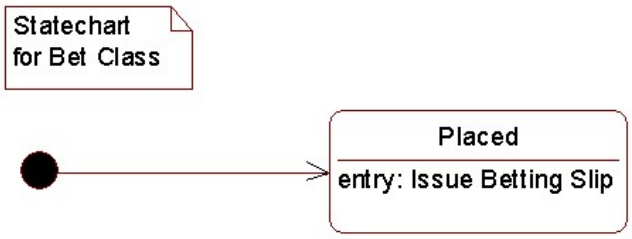

Figure 98 - Here, we need to issue a betting slip when the state change occurs

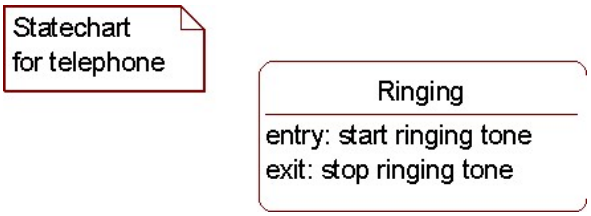

Figure 99 - Here, the ring tone starts on entry to the state - the ring tone stops on exit

Send Events

The above notation is useful when you need to comment that a particular action needs to take place. Slightly more formally, we can tie this approach to the idea of objects and collaboration. If a state transition implies that a message has to be sent to another object, then the following notation is used (alongside the entry or exit box):

```
^object.method (parameters)
```

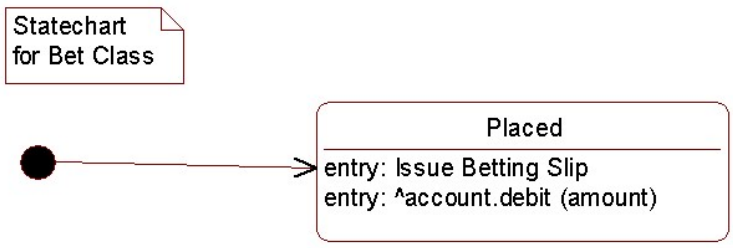

Figure 100 - formal notation indicating that a message must be sent on the state transition

Guards

Sometimes we need to insist that a state transition is possible only if a particular condition is true. This is achieved by placing a condition in square brackets as follows:

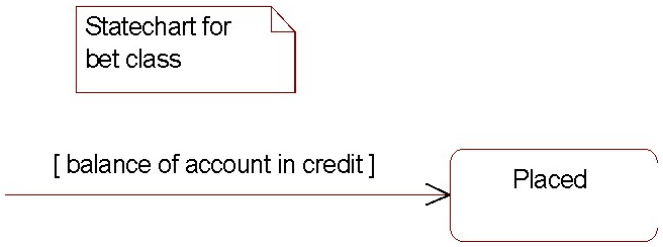

Figure 101 - Here, the transition to the "Placed" state can only occur if the balance of the account is in credit

History States

Finally, returning to substates briefly, it is possible to notate that if the superstate is interrupted in some way, when the superstate is resumed, the state will be remembered.

Take the following example. A criminal investigation starts in the "pending" state. Once it switches to the "Being Actioned" state, it can be in a number of substates.

However, at random intervals, the case can be audited. During an audit, the investigation is briefly suspended. Once the audit is complete, the investigation must resume at the state from where it was before the audit.

The simple "history notation" (a "H" in a circle) allows us to do this, as follows:

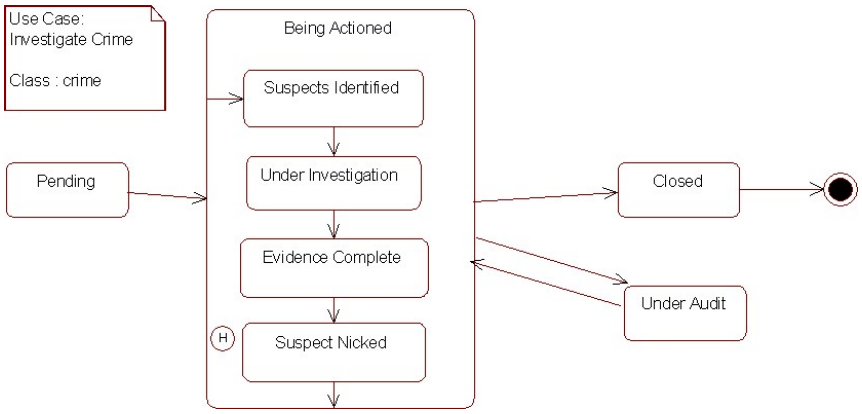

Figure 102 - History State

Other Uses for State Diagrams

Although the most obvious use for these diagrams is to track the state of an object, in fact, statecharts can be used for any state-based element of the system. Use Cases are a clear candidate (for example, a use might only be able to proceed if the user has logged on).

Even the state of the entire system can be modelled using the statechart - this is clearly a valuable model for the "central architecture team" in a large development.

Summary

In this chapter, we looked at State Transition Diagrams.

We saw:

- The syntax of the diagram
- How to use Substates
- Entry and Exit Actions
- Send Events and Guards
- History States

Statecharts are quite simple to produce, but often require deep thought processes

Most commonly produced for Classes, but can be used for anything : Use Cases, entire Systems, etc