Tutorial: state diagrams

October 5, 2012

Purpose

Let you practise developing state diagrams, first very simple ones and then more complex ones.

Exercises

Everyone should become able to do these. If you have trouble ask your tutor.

1. A party object is created when a venue is booked. An entertainer booking and then a food booking can be added. Before the party takes place a deposit of 20% must be paid. After it takes place, the remaining cost must be paid; once it has been paid, the party object is destroyed.

   Draw a protocol state diagram for the class Party to show the lifecycle of Party objects. Do not include conditions yet (notice whether this causes a problem). Do consider carefully what – in terms of the interface of Party, which you will need to enhance – causes each transition. Remember to include the `getTotalCost()` method that you considered in the Week 3 tutorial.

   Draw object diagrams to show how the configuration of objects in the system changes through a typical lifecycle, and list any changes you need to make to the interface of classes in your system, such as new operations for Party.

2. A system developed according to your protocol state diagram is built and deployed, but then the customer reports a catastrophic problem: they are receiving too little money and the system is not complaining! Some investigation reveals that the system is allowing your customer's clients to underpay both the deposit and the balance. Is this permitted by your model? If so, try to fix the problem (i.e., to reduce the likelihood that a system that satisfied your model could have this problem) using conditions on transitions as appropriate.

3. Suppose we wish to show that the party can be cancelled at any time before it takes place. We can use nested states to do this. Show how.

4. In a future release of the software, the processes of booking the entertainer and the food can be done in either order. Use concurrent subregions to show this.

5. What difference would it have made if you had been asked to draw behavioural state diagrams instead of protocol state diagrams? (Hint: not much.)

More challenging exercises

Do these now if you have time and are happy with the basic exercises above; otherwise, do them later for revision.

1. Consider a (toy) class `NaturalNumber` which contains an Integer attribute `i`, with the *class invariant* that `i` is always at least 0, and operations `increment()` and `decrement()` which generally have the obvious behaviour of incrementing and decrementing `i`; if `decrement()` is sent to an object of class `NaturalNumber` in which the value of `i` is 0, then instead of decrementing `i`, the object will send to an object `log` (to which it has a reference) the message `triedToDecrementZero()`.

   (a) Draw a behavioural state diagram for `NaturalNumber` that records all the behaviour described above, and has two states.

   (b) Now do the same, but make sure your diagram has three states.

   (c) Generalise... What choices do you have for the number of states in a correct state diagram for `NaturalNumber`? Can you have a correct behavioural state diagram that has one state?

   (d) Explain how this relates to the complete (fully-detailed) state space of `NaturalNumber`.

   (e) Draw a protocol state diagram for `NaturalNumber`, ensuring that you include all the information clients would need to use objects of this class correctly.

2. Vague question to think about: visibility of the operations and attributes that appear in a state diagram. Is it OK to mention private operations in a state diagram? Private attributes? What are the implications for good design?