Tutorial: conceptual modelling

September 30, 2012

Purpose

Let you practise conceptual modelling.

This tutorial is open-ended (much more so than, for example, an exam question could ever be). There aren't any right answers as such and there are different ways of proceeding. **It would be ideal to prepare this tutorial in groups of at least two people – doesn't matter whether the people are in the same tutorial group or not.** In the tutorial you'll discuss all the solutions generated by tutorial group members, especially, the ways in which they differ.

Exercises

Consider the following rough outline of system requirements, which you have received from the Managing Director of Party Planning Ltd.

> Our business is party organisation. We have good relationships with a number of venues, entertainers, caterers etc. in the area (Edinburgh city centre). Our clients come to us with ideas about how they want their party to be. We find the one fixed element is usually the date and time; beyond that, they usually want us to make suggestions. We get them to tell us their rough budget and have a conversation about their idea of a great party. We look at availability of venues, entertainers and caterers on their date, and put together usually two or three contrasting quotations for discussion. The client often combines elements of several quotes to arrive at a party plan that pleases them and that we can offer. At this stage the client signs a contract with us and pays a deposit, which is a fixed percentage of the entire party cost. We may or may not have to pay deposits in turn to the venues etc. The client gives us a list of invitees and we handle the sending out of invitations and the collection of replies. We'll have estimated how much food is required but we sometimes have to adjust in the light of how many invitees accept the invitation. We send the client the final list of acceptances. Occasionally something goes wrong at the last minute, for example, last week the Blue Hall flooded and we had to relocate all the parties we'd planned in it to other venues and send out notices to all the clients and party invitees concerned. We gave clients a discount to compensate them for the inconvenience. On the day casual employees of ours run the show at the venue, setting up, organising the caterers, and clearing up afterwards. The final bill to the customer is sent the morning after the party and includes charges for any breakages, unusual cleaning etc.
>
> Our business is highly seasonal and we have a high staff turnover in the office. We used to have one person with a memory like an elephant who could keep track of what needed to be done, but lately we've found things are getting forgotten. We'd like a software system that will keep track of where everything's at and help our staff know what they should do next. Otherwise our reputation will start to suffer; we're already finding our ratings falling in the questionnaires we send out with the bills.

1. Before you start looking at this as a SEOC exercise, look at it as a real business problem. What options should Party Planning's MD be considering, besides ordering a software system from you?

   From now on this is a SEOC problem and you're going to design an object oriented software system along the lines the customer envisages. Throughout, notice places where you would need more information than is above, and then feel free to invent the customer's answers to your questions.

2. Use noun identification to generate an initial list of classes. Notice the cases where a noun doesn't become a potential class and why. Are any clearly-needed classes missing? Put in relationships where you can.

3. Turn the description above into a first draft set of use case descriptions, summarised by a use case diagram. Identify a few use cases to be provided in Version 1 of the system.

4. Taking each use case in turn, use alternately robustness analysis and CRC cards (one for one use case, one for the next) to play through the use case (including any unusual scenarios). Revise the use case text and the conceptual class diagram as you go. Notice that there should always be **one** conceptual class diagram: changes you make for one use case alter the starting point from which you consider the next.

5. Pick a non-trivial scenario you understand well and draw a fully correct UML sequence diagram to describe it.

6. (Optional) Prototype your system in Java. Aim to check your understanding of the relationship between your modelling and the eventual system and of the system aspects that your modelling so far has not covered, rather than to produce a system you can actually start Party Planning with (although feel free to do that if you like!). For example, you might want to have your system interact with the user via simple text on stdout and stdin, rather than by a GUI.