Exercises on Use Cases

Wishnu Prasetya (wishnu@cs.uu.nl)

2010/11

1. Many people have cats. Suppose you want to develop software enabling them to keep track the location of their cats; give a *use case diagram* modeling this software. In this example, just decide by yourself what you want from this software.

   Theory questions:

   (a) What is the main purpose of use case modeling?

   Answer: To capture the software's functional requirement.

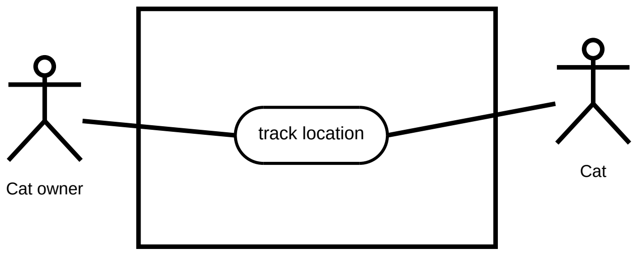
   (b) Give few examples of non-functional requirements that could be sensical for this software. How do you position use-case modeling with respect to such requirements?

   Answer: E.g. that the software should be platform independent (can run on Windows, Mac, Unix, etc); and that it should be web-browserwe-based. Such requirements cannot be captured with a use case and should thus be documented complementary.

   Modeling question: should 'cat' be an actor?

   Answer: Either way can. You can see 'cat' as a (secondary) actor that provides its location to your software (e.g. through some small hardware attached to it). Or you can see it as a part of the system.

   Answer:



Just a minimalistic one.

2. Work out at least one use case in your model above to the *casual* detail level.

   Work out your use-case further to the *fully dressed* level; while you do that, give an example of an alternate flow that would be sensical in your use-case.

   Should you just keep one model, or both models?

   Answer:

   Casual:

Use case: track location.

Primary actor: Owner

Secondary actors: Cat

The owner requests the locations of her cat(s). The system shows her their last known locations.

In order to know the locations, the system regularly checks the cats.

Fully dressed:

Use case: track location.

Primary actor: Owner

Secondary actors: Cat

Pre-condition: -

Post-condition: -

Main flow:

A1. The use-case is activated when the Owner requests it.

A2. The system checks the last known locations of the owner's cats.

A3. The system displays the location.

To keep track the locations, the system continously polls the cats in the background:

B1. loop forever:

  B2. for each cat, check the cat location. Update knowledge with this location.

Alternate flow:

A3b. If the last know location of a cat is timed has not changed since the last time the owner asked, and it is too old (> 3 hour), warn the owner.

---

3. Give situations (one for each) where it would be sensical to use these constructs to structure your use case above:

   - $\ll$ `include` $\gg$
   - $\ll$ `extend` $\gg$
   - specialization

   Answer:

   (a) E.g. if the polling in the example above is quite involved, we can separate it to its own use case, and link it with $\ll$ `include` $\gg$.

   (b) If the alternate flow in the example above becomes too large, we can separate it in its own use case, and link it with $\ll$ `extend` $\gg$.

   (c) Suppose we also provide tracking of wild animals, which could be a bit different than tracking a house cat. E.g. we need to poll less frequently. A use case for this can be made as a *specialization* from that of tracking cat.

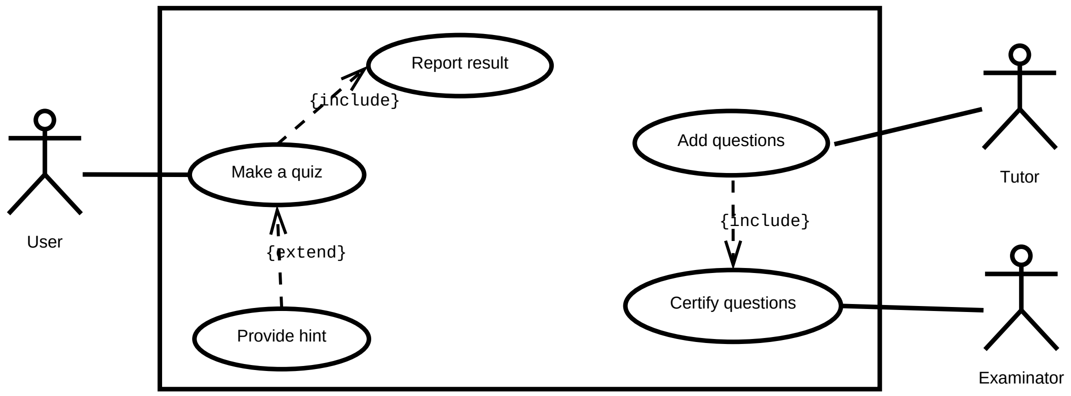
4. Do you know that it costs a lot of money to get a 'Certified Java Programmer' certificate? It could cost you thousands of euros. Let's imagine we will develop a browser-based training system to help people prepare for such a certification exam.

   A user can request a quiz for the system. The system picks a set of questions from its database, and compose them together to make a quiz. It rates the user's answers, and gives hints if the user requests it.

In addition to users, we also have tutors who provide questions and hints. And also examinators who must certify questions to make sure they are not too trivial, and that they are sensical.

Make a use case diagram to model this system. Work out some of your use cases. Since we don't have real stake holders here, you are free to fill in details you think is sensical for this example.

Answer:



We'll assume multiple choice quiz.

Use case: Make quiz.
Primary actor: User
Secondary actors: -
Pre-condition: The system has at least 10 questions.
Post-condition: -
Main flow:
1. The use-case is activated when the user requests it.
2. The user specifies the difficulty level.
3. The system selects 10 questions, and offers them as a quiz to the user.
4. The system starts a timer.
5. For every question:
   5a. The user selects an answer, or skip. [Extension point]
6. If the user is done with the quiz, or the timer runs out, the quiz is concluded, and [include use case 'Report result'].

Use case: Provide hint
Primary actor: User
Secondary actors: -
Pre-condition: The user requests for a hint.
Post-condition: -
Main flow:
1. The system provides a hint. The verbosity of the hint is determined by the difficulty level set previously by the user.
2. Return to to Make quiz' main flow.

---

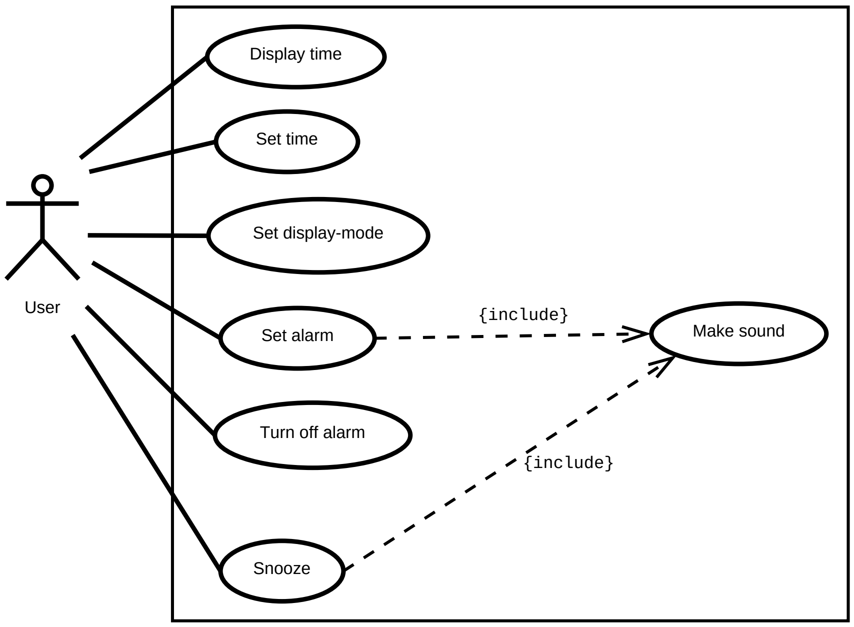
5. Suppose we want to develop software for an alarm clock.

   The clock shows the time of day. Using buttons, the user can set the hours and minutes fields individually, and choose between 12 and 24-hour display.

   It is possible to set one or two alarms. When an alarm fires, it will sound some noise. The user can turn it off, or choose to 'snooze'. If the user does not respond at all, the alarm will turn off itself after 2 minutes. 'Snoozing' means to turn off the sound, but the alarm will fire again after some minutes of delay. This 'snoozing time' is pre-adjustable.

   - Identify the top-level functional requirement for the clock, and model it with a use case diagram.

     Answer:



     In this model 'make sound' is made as a separate use case. It does not have to.

   - I assume that 'snooze' would be one of your use cases. Work it out to the fully dressed level.

     Answer:

Use case: Snooze.
Primary actor: User
Secondary actors: -
Pre-condition: An alarm is firing.
Post-condition: -
Main flow:
1. The use-case is activated when the user hits the snooze button.
2. The alarm is turned off.
3. Wait for snooze time.
4. Include use case 'Make sound'

Use case: Make sound
Primary actor: System
Secondary actors: -
Pre-condition: -

Post-condition: -
Main flow:
The use case starts when it is called. What it does is to just make some noisy sound.
