1.8 Supplement: A Brief Tutorial on Writing Use Cases

Anything that has behavior is an *actor.* This convention allows us to refer equally easily to people, computer programs, and companies, without worrying about which category of actor is playing the role at that moment. A use case, then, describes the way in which a particular system under discussion (SuD), an actor in its own right, interacts with other actors.

To describe the many complicated interactions that a system will have over its lifetime, we link any one use case with the goal of an actor who wants something from the SuD at that moment, and describe all the ways that the system may come to deliver or abandon the goal of that "primary actor."

Then we structure the writing in an interesting way: First of all, we describe how the actors behave in a simple situation in which the goal gets achieved. After that, we name all the conditions under which the behavior is different, and describe the different behavior that ensues, always bearing in mind that sometimes the goal will succeed and sometimes it will fail. These are called *extensions* or *alternate courses* of behavior within the use case.

We can see now that the use cases discussed so far were just fragments, since they described only the simple case of goal success (what some people call the "happy day" scenario). The complete use case is too long to put in here, but looks essentially like Use Case 1.3.

Use Case 1.3 *Register for Courses:* Use Case with Extensions

Register for Courses (Use Case with Extensions)

Primary actor: Student

System under Discussion: Course Enrollment System

Level: User Goal

1. Student requests to construct a schedule.
2. The system prepares a blank schedule form.
3. The system pulls in a list of open and available courses from the Course Catalog System.
4. Student selects up to 4 primary course offerings and 2 alternate course offerings from the available offerings.
5. For each course, the system verifies that the Student has the necessary prerequisites and adds the Student to the course, marking the Student as "enrolled" in that course in the schedule.
6. When the Student indicates the schedule is complete, the system saves the schedule.

Extensions:

1a. Student already has a schedule:

System brings up the current version of the Student's schedule for editing instead of creating a new one.

1b. Current semester is closed and next semester is not yet open:

System lets Student look at existing schedules, but not create new ones.

3a. Course Catalog System does not respond:

The system notifies the Student and terminates the use case.

4a. Course full or Student has not fulfilled all prerequisites:

System disables selection of that course and notifies the Student.

People sometimes find that a briefer way of writing is desirable (for example, for very small projects, and projects in which the use cases are merely being used to estimate work effort, not specify the system). In other situations, a more exact way of writing is needed (such as military contract outsourcing, large distributed projects, and life-critical systems). It is important to recognize that there is no one, best format for use cases, but that the amount of detail to put into a use case depends on the project and the team at hand, and the purpose of use case writing.

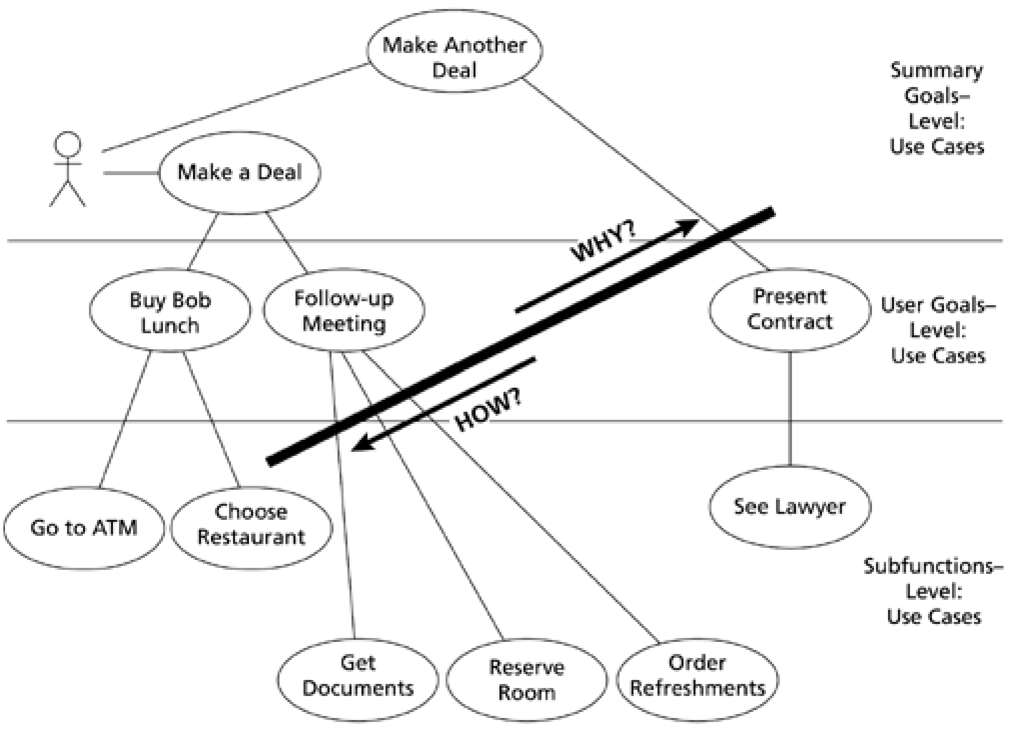
While it is all very well to say that a use case describes the behavior involved in trying to achieve a goal, the difficulty is that goals exist at many levels. Any one goal is achieved by achieving subgoals. For example, I might have a goal to send my children to school in a rich section of the city. To achieve that goal, I have to earn enough money to buy a house in that school district. To do that, I need to close some big business deals, so my next subgoal is to win some particular contract. To do that, I may decide my next subgoal is to win over a particular decision maker, and so I take him to lunch for a discussion.

Each of these goals can be described using a use case, although I have not yet specified a particular SuD for them. Continuing the subgoals, I find I need cash to take him to lunch, and so I go to a local cash-dispensing machine, where my goal is to get some cash. My first subgoal, now directly related to the cash machine as an SuD, is to identify myself, to which end I have subgoals to insert my card into the machine, type in my identification number, and hit the Enter key. (People who know Alistair have already learned that he can make almost any goal involve going to an ATM to get cash!)

All in all, I could write any of the following use cases within the framework of the information given so far: *Find Enter Key, Authorize User, Insert ATM Card, Get Cash, Win Contract, Buy a Too-Expensive House, Get Kids into Good School.*

This capacity of use cases to describe goals at all levels is wonderful but confusing to use case writers. Different writers describe different levels of goals, some staying high *(Get Cash),* and some staying at terribly low levels *(Authorize User, Insert ATM Card).* For most systems, it is critical to identify the goal level in which the system contributes direct, take-home value to the primary actor (see the entry UserValuedTransactions (p. 95). This level we refer to as *user goal*. Then we write additional use cases for goals at higher and lower levels as needed. These we refer to as *summary* and *subfunction,* respectively. Figure 1.2, adapted from *Writing Effective Use Cases* (Cockburn 2001), illustrates goal levels by describing some of the deals necessary to get the kids into a better school.

Figure 1.2. Goal levels for sending your kids to a better school



Use cases are read and used by two very different groups of people: (1) end users or business experts, who often are not versed in the technical and implementation difficulties; and (2) programmers, who need very precise answers to their questions in order to make the computer program work properly. It is not obvious that any form of writing can satisfy both groups of people, but use cases have shown themselves as fitting this narrow range. The art of use case writing is to get the precision in writing without overwhelming the non-programmer business reader.

To create use cases that are correct and precise but still readable, the writing team *must* include:

- At least one person with a background in programming, to get the required accuracy and precision of description
- At least one person with deep knowledge of the business rules that the system must enforce
- At least one person with intimate knowledge of how the system will actually be used

In other words, producing a set of use cases is not a job for one person, or even one group of people with the same job description. It is a team effort, requiring people with different backgrounds and even different personalities. When this team does its job well, the result is readable *and* precise.

This book is not an introduction to use cases. Rather, it is a handbook about how to write meaningful, high-quality use cases. Therefore, we have provided only a brief summary of use cases in this chapter. If you want to learn more about use cases in general, we recommend Alistair Cockburn's *Writing Effective Use Cases* (2001). More discussion of use cases is available at the Web site www.usecases.org. You may refer to these sources for introductory, template, and tools-descriptive material, and continue reading this book to improve your ability to detect and discuss the signs of quality in your use case writing.

I l@ve RuBoard PREVIOUS NEXT