Dear Dr. Use Case: *Is the Clock an Actor*?

by **Anthony Crain**
Software Engineering Specialist
Rational Software

Dear Dr. Use Case,

I have a question about using a nonhuman trigger as the Primary Actor of a use case. For example, I have seen *System Clock* used as a Primary Actor. Is this really the best way to model a use case that is triggered by time?

Signed,
Wondering about Nonhuman Actors

Dear Wondering,

This is an excellent question, and one that I have been asked many times before. When I first started modeling use cases, I tried showing the clock as a primary actor to initiate use cases that are triggered automatically. Since then, though, my experience has shown me a better way.

Let me illustrate by borrowing an example use case called Run Payroll from our Rational University course *Object Oriented Analysis and Design with the UML (OOAD)*. Here's the brief description given in the courseware: "This use case describes how the payroll will be run automatically every Friday and the last working day of the month." Originally, this use case had *System Clock* as its Primary Actor (a.k.a. the actor that wanted to run the payroll). See Figure 1.

Figure 1: System Clock as the Primary Actor in the Run Payroll Use Case

The first thing that bothered me about this was that System Clock was the Primary Actor. In truth, the system clock is actually a design decision and not really an actor. Fundamentally, it is just one way of capturing time, so it's more accurate to think of the Primary Actor as Time rather than System Clock (see Figure 2).

Figure 2: Time as the Primary Actor in the Run Payroll Use Case

Primary Actors Have System-Related Goals

It's also important to understand that a Primary Actor has goals relating to the system we are trying to build. If a student has a goal of "register for courses" or "view report card," for example, and our system supports these goals, then we can transform these goals into use cases, with the student as the Primary Actor (see Figure 3)..

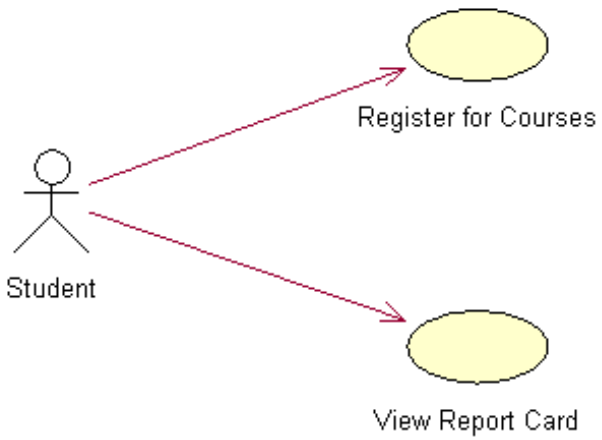

Figure 3: Transforming Goals into Use Cases

When I see an actor whose <<communicates>> association (the only kind

allowed on a use-case model diagram) points at a use case, I interpret that to mean that the use case is that actor's goal.

So suppose that *Time* is really the Primary Actor for Run Payroll. Then we can assume that Time has the goal of running payroll. Does that ring true? Well, let's look at it another way for a moment. One could say that Time has many other goals: Wither the young, erode continents, heal all wounds, and so on. But since these goals do not intersect with what our system provides, none of them would translate into a use case for our system. Run Payroll, however, is a goal Time has that intersects with our system, so therefore it is a legitimate use case (see Figure 4).

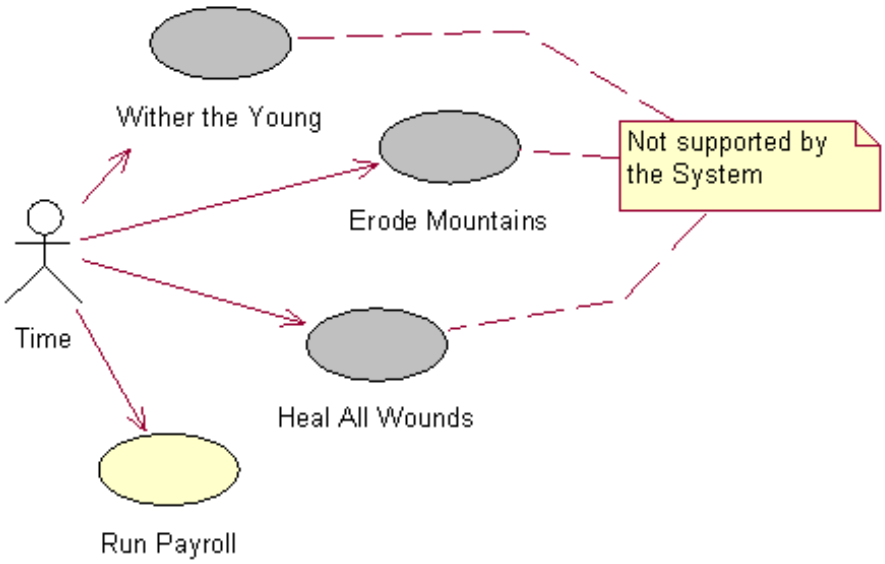

Figure 4: Determining What Goals Intersect with the System

Should we accept this logic? If we do, then suddenly, any nonhuman trigger can be the Primary Actor for a use case: Humidity, Dew Point, Temperature, Light, and more! At first, this concept seemed so powerful to me that I felt it just couldn't be wrong.

A Problem for Black Box Use Cases

One thing about it plagued me, however. The technique I use for writing use-case scenarios and flows is to discuss only the *visible* behavior of a system. This is called a black-box use case. But to what (or whom) is that behavior visible? To any actor that touches the use case. So for Run Payroll, if Time starts the use case, then that's it for the visible behavior! By extension, whenever an *event* triggers a use case, then, by definition, the use-case specification will be trivial. When I realized this, I stopped using triggers as the Primary Actors for my use cases.

In some instances, however, a use case has one or more Secondary (or *Supporting* as they say in Hollywood) Actors, in which case it might have some meat to it.

Another approach is to create white-box use cases that model internal

behavior. But I find that this makes use cases very hard for the customer, black-box functional tester, technical writer, user interface designer, and so on, to understand. Instead, I model the internal behavior as rules in a separate artifact that I call a Rules Dictionary. Also, white-box use cases are likely to incorporate too much design; black-box use cases with a supporting rule set tend to be better focused on requirements.

A Human Alternative

So what is the best alternative? In our example, who *really* has the goal of running payroll? Well, if we look back at the use-case model diagram in the *OOAD* course, we see that there is a Payroll Administrator, a person or team responsible for ensuring that employees are paid. Remember: A system automates a business process, and in this case, we are automating the business process of the Payroll Administrator. Aha! So then it is the Payroll Administrator who has the goal of running payroll.

Now, the question is: Why did the analyst use Time as the Primary Actor, instead of the Payroll Administrator? Mainly, to avoid implying that the Payroll Administrator must manually start the payroll processing. But keep in mind that, if we use the Payroll Administrator as the Primary Actor, then we can capture all the system features that surround the running of payroll. This includes features that allow the Payroll Administrator to set the timetables for running payroll and to handle discrepancies, manual intervention, and holidays. And since much of this would be *visible* behavior, it would fit nicely into the use case.

Handling the Time Dependency

Using the Payroll Administrator as the Primary Actor also gives us two ways to handle the time dependency in the use case:

1. **Set Time as a Secondary Actor.** Some people choose to portray time as a Secondary Actor (see Figure 5). This shows that time is a factor in accomplishing the use case. A nice side effect is that if you follow the Boundary, Entity, Control pattern for analysis suggested in *OOAD* and in the Rational Unified Process®, the analysis model will then contain a <<boundary>> class to time. *This method is best if it is important to show which use cases have a time dependency.*

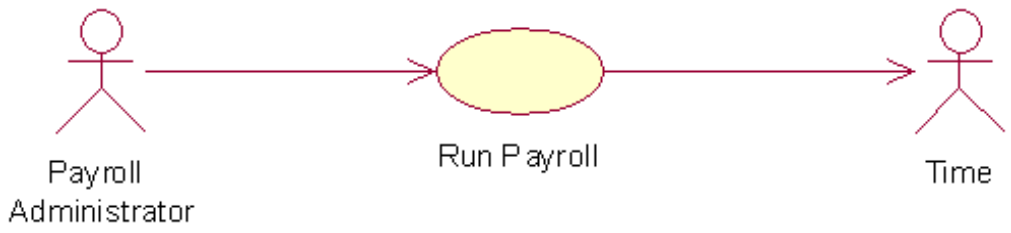

Figure 5: Payroll Administrator as the Primary Actor in the Run Payroll Use Case

2. **Create a Mechanism to Capture Time.** A second method is to create a mechanism to capture time. With this method, there is nothing in the use-case model diagram to indicate which use cases

are time triggered (see Figure 6), but we can still determine that by examining the Use-Case View of Architecture. This also puts finding a solution to the problem squarely in the Architect's court.

Figure 6: Abstracting Time and Placing the Problem in the Architect's Court

Your Choice

So, in short, my personal answer to the question, "Can the system clock be the Primary Actor of a use case?" is "no." However, there is nothing in the Unified Modeling Language (UML) that says it can't; and if you typically write white-box use cases, then maybe your guidelines will point to "yes." My advice is to ask yourself: Who truly wants the functionality? Then designate that person as the Primary Actor, and leave design intricacies such as capturing time in the application to the Architect and his or her mechanisms.

Usefully yours,
Dr. Use Case

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!