Introduction to UML sequence diagrams

Modeling the logic of use cases

This introduction to the Unified Modeling Language (UML) notation for sequence diagrams has been adapted from Chapter 6 of *The Object Primer 2nd Edition*.

PDF (144 KB) | 0

Comments

Share:

Scott W. Ambler, Practice Leader, Agile Development, Rational Methods Group, IBM, Software Group

11 January 2001

Also available in Japanese

- Table of contents

Sequence diagrams are used to model the logic of usage scenarios. A usage scenario is exactly what its name indicates -- the description of a potential way your system is used. The logic of a usage scenario may be part of a use case, perhaps an alternate course. It may also be one entire pass through a use case, such as the logic described by the basic course of action or a portion of the basic course of action, plus one or more alternate scenarios. The logic of a usage scenario may also be a pass through the logic contained in several use cases. For example, a student enrolls in the university, and then immediately enrolls in three seminars. Sequence diagrams model the flow of logic within your system in a visual manner, enabling you to both document and validate your logic, and are commonly used for both analysis and design purposes.

Figure 1 models the basic course of action for the "Enroll in Seminar" use case. You may want to open the figure now and refer to it as you read this tip.

Classifiers

The boxes across the top of the diagram represent classifiers or their instances -- typically use cases, objects, classes, or actors (usually depicted as rectangles, although they can also be symbols).

Because you can send messages to both objects and classes (objects respond to messages through the invocation of an operation and classes do so through the invocation of static operations), it makes sense to include both on sequence diagrams. Because actors initiate and take an active part in usage scenarios, they are also included in sequence diagrams. Objects have labels in the standard UML format "name: ClassName," where "name" is optional. (Objects that haven't been given a name on the diagram are called anonymous objects.) Classes have labels in the format "ClassName," and actors have names in the format "Actor Name" -- both UML standards as well.

For example, in Figure 1, you see the "Student" actor has the name "A Student" and is labeled with the stereotype <<actor>>. The instance of the major UI element representing "UI32 Seminar Selection Screen" is an anonymous object with the name ":SeminarSelector" and the stereotype <<UI>>. The "Student" class is indicated on the diagram (the box with the name "Student") because the static message "isEligible(name, studentNumber)" is sent to it. More on this later.

In the diagram, the instance of "Student" was given a name "theStudent" because it is used in several places as a parameter in a message. Contrast this with the instance of the "StudentsFees" class, which didn't need to be referenced anywhere else in the diagram and, thus, could be anonymous.

Lifelines

The dashed lines hanging from the boxes are called object lifelines, representing the life span of the object during the scenario being modeled. The long, thin boxes on the lifelines are method-invocation boxes that indicate processing is being performed by the target object/class to fulfill a message. The X at the bottom of a method-invocation box is a UML convention to indicate an object has been removed from memory, typically the result of receiving a message with the stereotype of <<destroy>>.

Modeling messages

Messages are indicated as labeled arrows. When the source and target of a message is an object or class, the label is the signature of the method invoked in response to the message. However, if either the source or target is a human actor, then the message is labeled with brief text describing the information being communicated. For example, the ":EnrollInSeminar " object sends the message "isEligibleToEnroll(theStudent)" to the instance of "Seminar. " Notice how I include both the method's name and the name of the parameters, if any, passed into it.

Figure 1 also indicates that the "Student" actor provides information to the ":SecurityLogon " object via the messages labeled "name" and "student number." (These really aren't messages; they are actually user interactions.) Return values are optionally indicated using a dashed arrow with a label

indicating the return value. For example, the return value "theStudent" is indicated coming back from the "Student" class as the result of invoking a message, whereas no return value is indicated as the result of sending the message "isEligibleToEnroll(theStudent) " to "seminar." My style is not to indicate the return values when it's obvious what is being returned, so I don't clutter my sequence diagrams. (As you can see, sequence diagrams get complicated fairly quickly.)

Messages fulfill the logic of the steps of the use case, summarized down the left-hand side of the diagram. Notice how the exact wording of the use-case steps isn't used because the steps are often too wordy to fit nicely on a diagram. What is critical is that the step numbers correspond to those in the use case and the general idea of the step is apparent to the reader of the diagram.

Figure 1 shows a UML sequence diagram for the basic course of action for the Enroll in Seminar use case listed below.

The "Enroll in seminar" use case Name: Enroll in SeminarIdentifier: UC 17Description: Enroll an existing student in a seminar for which he is eligible.Preconditions: The Student is registered at the University.Postconditions: The Student will be enrolled in the course he wants if he is eligible and room is available.Extends: N/AIncludes: N/AInherits From: N/A**Basic course of action:**

1. The student wants to enroll in a seminar.
2. The student inputs his name and student number into the system via "UI23 Security Login Screen."
3. The system verifies that the student is eligible to enroll in seminars at the university, according to business rule "BR129 Determine Eligibility to Enroll."
4. The system displays "UI32 Seminar Selection Screen," which indicates the list of available seminars.
5. The student indicates the seminar in which he wants to enroll.
6. The system validates that the student is eligible to enroll in the seminar, according to the business rule "BR130 Determine Student Eligibility to Enroll in a Seminar."
7. The system validates that the seminar fits into the existing schedule of the student, according to the business rule "BR143 Validate Student Seminar Schedule. "
8. The system calculates the fees for the seminar based on the fee published in the course catalog, applicable student fees, and applicable taxes. Apply business rules "BR 180 Calculate Student Fees" and "BR45 Calculate Taxes for Seminar."
9. The system displays the fees via "UI33 Display Seminar Fees Screen."
10. The system asks the student whether he still wants to enroll in the seminar.
11. The student indicates that he wants to enroll in the seminar.
12. The system enrolls the student in the seminar.
13. The system informs the student the enrollment was successful via "UI88 Seminar Enrollment Summary Screen."
14. The system bills the student for the seminar, according to business rule 'BR100 Bill Student for Seminar."
15. The system asks the student if he wants a printed statement of the enrollment.
16. The student indicates that he does want a printed statement.
17. The system prints the enrollment statement "UI89 Enrollment Summary Report."

18. The use case ends when the student takes the printed statement.

Alternate course A: The student is not eligible to enroll in seminars

1. The system determines the student is not eligible to enroll in seminars.
2. The system informs the student he is not eligible to enroll.
3. The use case ends.

Alternate course B: The student does not have the prerequisites

1. The system determines that the student is not eligible to enroll in the seminar he has chosen.
2. The system informs the student that he does not have the prerequisites.
3. The system informs the student of the prerequisites he needs.
4. The use case continues at Step 4 in the basic course of action.

Alternate course C: The student decides not to enroll in an available seminar

1. The student views the list of seminars and doesn't see one in which he wants to enroll.
2. The use case ends.

Resources

- *The Object Primer 2nd Edition* by Scott W. Ambler. New York: Cambridge University Press, 2001.
- *The Unified Process Inception Phase* by Scott W. Ambler and Larry L. Constantine. Gilroy, CA: R&D Books, 2000.
- *The Unified Modeling Language Reference Manual* by James Rumbaugh, Grady Booch, and Ivar Jacobson. Reading, MA: Addison-Wesley Longman, Inc., 1999.