Web services programming tips and tricks: Modeling essential use cases

Where you're coming from, and where you're going

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

20 July 2000

Essential modeling is a fundamental aspect of usage-centered designs. This week Scott Ambler presents some background and suggestions for developing essential use case models.

An important goal in requirements modeling is to come to an understanding of the business problem that your system is to address, in order to understand its behavioral requirements. With respect to object-oriented development, the fundamental artifact that you should develop to model behavioral requirements is a *use case model*. Use case diagrams are among the standard Unified Modeling Language (UML) artifacts. There are two basic flavors of use case models: *essential* use case models, and *system* use case models.

- An essential use case model, often referred to as a *business* or *abstract* use case model, models a technology-independent view of your behavioral requirements.
- System use case models, also known as *concrete* use case models or *detailed* use case models, model your analysis of your behavioral requirements, describing in detail how users will work with your system including references to its user-interface aspects.

What is essential modeling?

Essential modeling is a fundamental aspect of usage-centered designs -- an approach to software development that is detailed in the book *Software for Use* (see Resources). Essential models are intended to capture the essence of problems through technology-free, idealized, and abstract descriptions. The resulting design models are more flexible, leaving more options open and more readily accommodating changes in technology. Essential models are more robust than concrete representations simply because they are more likely to remain valid in the face of changing requirements and changing implementation technology. Essential models of usage highlight

purpose, what it is that users are trying to accomplish, and why they are doing it. In short, essential models are ideal artifacts to capture the requirements for your system.

What is a use case?

A use case is a sequence of actions that provide a measurable value to an actor. Another way to look at it is that a use case describes a way in which a real-world actor interacts with the system. An essential use-case is a simplified, abstract, generalized use case that captures the intentions of a user in a technology- and implementation-independent manner. An essential use case is a structured narrative, expressed in the language of the application domain and of users, comprising a simplified, generalized, abstract, technology-free and implementation-independent description of one task or interaction. An essential use case is complete, meaningful, and well designed from the point of view of users in some role or roles in relation to a system and that embodies the purpose or intentions underlying the interaction.

Comparing a system use case and an essential use case

Consider the two versions of the use case "Enroll in seminar" presented in Example 1, which presents a simplified system use case (also called a traditional or concrete use case), and in Example 2, which presents it as an essential use case. Some interesting observations:

1. The system use case has many implementation details embedded within it. For example, the concept of a registrar disappeared and was replaced with the term *system*, indicating that a decision had been made to automate many of the mundane aspects of enrollment. The writer of system use cases is analyzing and describing requirements imposed by the problem intermingled with implicit decisions about what the user interface is going to be like; the writer of an essential use case does not.
2. The system use case makes references to screens and reports, such as "UI23 Security Login Screen" and "UI89 Enrollment Summary Report." The essential use case does not make these references. Once again, this reflects implementation details; someone has decided that the system will be implemented as screens (as opposed to HTML pages, perhaps) and printed reports. However, the essential use case could just as easily have referred to major user interface elements, the essential version of screens and reports, and, to tell you the truth, this is a practice that I recommend. I did not include references to UI elements in Example 2 in order to provide you with an example in which this has not been done.
3. Both versions reference business rule definitions, such as "BR129 Determine Eligibility to Enroll," because business rules reflect essential characteristics of your domain that your system must implement.
4. The system use case has more steps than the essential use case. This, in fact, reflects my style of writing use cases; I believe that each use case step should reflect one step only. There are a couple of advantages to this approach: the use case becomes easier to test because each statement is easier to understand and to validate; and alternate courses are easier to write because it is easier to branch from a statement when it does one only thing.
5. The use case steps are written in the active rather than passive voice. The statement "The registrar informs the student of the fees" is more succinct than the statement "The student is informed of the fees by the registrar."

6. Both versions end with a step like "The use case ends" or "The use case ends when..." to indicate that the logic for the course of action has been completely defined.

Traditional or system use cases typically contain too many built-in assumptions, often hidden or implicit, about the underlying technology implementation and the user interface that is yet to be designed. This is a good feature during your analysis and design efforts, but not so good for your requirement efforts. An essential use case, on the other hand, is based on the purpose or intentions of a user, rather than on the concrete steps or mechanisms by which the purpose or intention might be carried out. Write essential use cases as part of your required engineering efforts and then evolve them into system use cases during your analysis and design efforts.

Example 1: "Enroll in Seminar" as a system use case

Name: Enroll in Seminar

Description: Enroll an existing student in a seminar for which they are eligible.

Preconditions: The student is registered at the university.

Postconditions: The student will be enrolled in the course they want if they are eligible and there is room available.

Basic course of action:

1. A 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 that he wants to enroll in.
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 would still like 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 that 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 would like a printed statement of the enrollment.
16. The student indicates he would like 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.

Example 2: "Enroll in Seminar" as an essential use case

Name: Enroll in Seminar

Description: Enroll an existing student in a seminar for which they are eligible.

Preconditions: The student is registered at the university.

Postconditions: The student will be enrolled in the course they want if they are eligible and there is room available.

Basic course of action:

1. A student wants to enroll in a seminar.
2. The student submits his name and student number to the registrar.
3. The registrar verifies that the student is eligible to enroll in seminars at the university according to business rule "BR129 Determine Eligibility to Enroll."
4. The student indicates, from the list of available seminars, the seminar that he wants to enroll in.
5. The registrar 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."
6. The registrar validates that the seminar fits into the existing schedule of the student according to the business rule "BR143 Validate Student Seminar Schedule."
7. The registrar 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."
8. The registrar informs the student of the fees.
9. The registrar verifies that the student still wants to enroll in the seminar.
10. The student indicates he wants to enroll in the seminar.
11. The registrar enrolls the student in the seminar.
12. The registrar adds the appropriate fees to the student's bill according to business rule "BR100 Bill Student for Seminar."
13. The registrar provides the student with a confirmation that he is enrolled.
14. The use case ends.

Resources

- *Applying Use Cases: A Practical Guide* by G. Schneider and J.P. Winters
- *Use Case Driven Object Modeling with UML: A Practical Approach* by Doug. Rosenberg and Kendall Scott
- *The Object Primer 2 nd Edition* by Scott W. Ambler
- *The Unified Modeling Language Reference Manual* by James Rumbaugh, Grady Booch, and Ivar Jacobson
- *Software For Use: A Practical Guide to the Models and Methods of Usage-Centered Design* by Larry L. Constantine and Lucy A.D. Lockwood
- *Building Object Applications That Work: Your Step-By-Step Handbook for Developing Robust Systems with Object Technology* by Scott W. Ambler
- *Software Requirements* by Karl Wiegers

About the author

Scott W. Ambler

Scott W. Ambler is President of Ronin International, a consulting firm specializing in object-oriented software process mentoring, architectural modeling, and Enterprise JavaBeans (EJB) development. He has authored or co-authored several books about object-oriented development, including the recently released *The Object Primer 2nd Edition*, which covers, in detail, the subjects summarized in this article. He can be reached at scott.ambler@ronin-intl.com and at his Web site at www.ambysoft.com.