Use Cases and the UML

Brian Dobing
University of Lethbridge, Canada

Jeffrey Parsons
Memorial University of Newfoundland, Canada

INTRODUCTION

The Unified Modeling Language, or the UML (Booch, Jacobson & Rumbaugh, 1999), has rapidly emerged as a standard language and notation for object-oriented modeling in systems development, while the accompanying Unified Software Development Process (Jacobson, Booch & Rumbaugh, 1999) has been developed to provide methodological support for application of the UML in software development. The UML is a non-proprietary modeling language managed by the Object Management Group, a not-for-profit consortium, which also manages several related modeling specifications. The UML has evolved from its initial version, with UML 2.0 formally adopted by the OMG in June 2003. This article is based on the UML 1.5 specifications (OMG, 2003), as those for UML 2.0 have not been finalized. However, the role of use cases appears to be essentially unaffected by the changes proposed for UML 2.0.

The term "use case" was introduced by Jacobson (1987) to refer to text document that outlines "a complete course of events in the system, seen from a user's perspective" (Jacobson, Christerson, Jonsson & Overgaard, 1992, p. 157). The concept resembles others being introduced around the same time. Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen (1991), Wirfs-Brock, Wilkerson, and Wiener (1990), and Rubin and Goldberg (1992) use the terms "scenario" or "script" in a similar way. While use cases were initially proposed for use in object-oriented analysis and are now part of the UML, they are not inherently object-oriented and can be used with other methodologies.

BACKGROUND

A use case should have a clear goal and describe what should typically happen (but not how it should happen). Common examples would include a customer renting a video, purchasing an item, withdrawing funds from an account, and so forth. The use case also identifies the main "actors" involved, which, in these examples, could include the customer, employees (e.g., rental clerk), a device (bank machine), and so forth.

The use case differs from typical structured requirements analysis tools that preceded it in two important ways. First, the use case is largely text-based. Some refer to them as use case narratives to clearly distinguish them from use case diagrams, which provide an overview of the use cases and actors in the system. Together, they form the use case model. Structured analysis emphasized the importance of graphical tools, such as work flow and data flow diagrams. The UML has not abandoned diagrams; 13 are now included. The class, activity, communication (previously collaboration), sequence, state machine (previously statechart) and use case diagrams play important roles. But use cases are written in the customer's language, so that "users and customers no longer have to learn complex notation" (Jacobson et al., 1999, p. 38).

Second, use cases focus on complete transactions from the user's perspective. In particular, use cases have a goal, which comes from the goals of those who will be using the system (Cockburn, 2001). This also helps facilitate communication with the system's intended users. In UML terminology, a use case is initiated by an actor, usually a person in a particular role (e.g., cashier), but sometimes an external system. Other actors may be involved as well (e.g., customer). The use case provides a complete view of the transaction, from initiation to achievement of the defined goal.

Consistent with an object-oriented approach, use cases can also have generalizations and include/extend relationships. As with classes, generalization allows a child use case to override the behavior of its parent use case in certain situations. An include relationship is generally used when the same steps are required by several use cases, in order to avoid repetition. An included use case is dependent on base use cases and "never stands alone" (Booch et al., 1999, p. 221), although not everyone follows this convention. An extend relationship exists when a base use case incorporates another use case depending on certain conditions.

As discussed extensively in the next section, the content and format of use cases vary widely among published books and articles. In addition to the basic narrative, use cases may contain initial sections that specify assumptions, preconditions (that must be in place before the use case is applicable), and triggers that initiate the use case. At the conclusion, there may be specified postconditions (that must be true when the use case ends). Exceptions (or alternative paths) may also be documented along with relevant business rules that govern behavior within the use case. None of these are part of the UML 1.5 specifications (OMG, 2003), which contains no sample use cases or suggestions on format, but some or all of them may be useful in different situations.

ROLE OF USE CASES

Use cases have been all but universally embraced in object-oriented systems analysis and development books written since Jacobson et al. (1992). Despite this strong endorsement, there are many variations on Jacobson's original theme. First, there is a difference in content. Use cases, at least during the analysis phase, are generally viewed as a conceptual tool. The use case should emphasize "what" and not "how" (Jacobson et al., 1994, p. 146). This suggests use cases should not mention technology (e.g., Evans, 1999).

A review of use case examples shows that determining when the "what" ends and the "how" begins is not always easy. Brown (2002) interprets "what" to mean what the system will do rather than the internal implementation. Thus, his use cases include references to screen designs. So do those of Satzinger and Orvik (1996, p. 126). Others, such as Harmon and Watson (1998, p. 121) refer to specific technology (salesperson's laptop). And even Jacobson et al. (1992, p. 162) refer to a display "panel," "receipt button" and "printer" in one of their examples. Some use cases also include more detail on business rules. For example, the IBM Object-Oriented Technology Center (1997, p. 489) video store example includes the condition that customers who are not members pay a deposit of $60.00. In contrast, Kulak and Guiney (2000, p. 23, emphasis at source) state that use cases "should show the *what* exclusively," and their examples seem to follow this philosophy. However, as Larman (2002, p. 75) notes, use cases are not tied to object-oriented methodologies and thus are technology-independent in that sense.

Second, there are several variations proposed for use case formats. While the first use cases in Jacobson et al. (1992) were written as a paragraph of text, most others have adopted numbered steps. Soon after, Jacobson et al. (1994, p. 109) did so as well.

Third, the granularity of use cases also varies from coarse (few use cases) to fine (many). Most take a minimalist approach. Jacobson et al. (1994, p. 105) suggest that use cases should offer "measurable value to an individual actor". MacMaster (1997) argues that use cases be used only for main system functions. But White (1994, p. 7) states that "the collected Use Cases specify the complete functionality of the system". While Dewitz (1996) uses 11 use cases in her video store example, the IBM object-oriented technology center (1997) has 24. Kulak and Guiney (2000, p. 37) suggest that "most systems would have perhaps 20 to 50 Use Cases and some small systems even fewer". But, as they later point out (p. 88), "there are no metrics established to determine correct granularity". In contrast, Armour and Miller (2001, p. 244) claim that large systems may have hundreds of use cases.

Fourth, the level of detail within each use case also varies. For example, both Kulak and Guiney (2000, p. 125) and Armour and Miller (2001, p. 125) recommend limiting the length of the flow of events to two pages of text, but the latter also note that some practitioners prefer a few longer use cases to many short ones. Constantine and Lockwood (2000) distinguish between "essential" use cases, containing few if any references to technology and user interface implementation, and "concrete" use cases that specify the actual interactions.

Jacobson et al. (1999) advocate an iterative development approach in which both the number of uses cases and their level of detail increase as the life cycle progresses. They suggest that only the most critical use cases (less than 10%) be detailed in the first (inception) phase. As analysis progresses and requirements become firmer, additional use cases can be added and each can be expanded to include considerably more detail. The analyst could move toward concrete use cases or simply expand the detail within essential use cases. Some authors have become quite specific in describing the different levels. For example, Kulak and Guiney (2000) have identified four levels. However, knowing where to start, how far to go at each phase, and when to stop are clearly critical issues not easily resolved.

To further complicate the issue, some of those who favor fewer or less detailed use cases supplement them with "scenarios". Booch et al. (1999, p. 225) define scenarios as "basically one instance of a use case". "Add a customer" is a use case. Adding a specified customer with a particular name, address, and so forth is a scenario. Some references use scenarios to provide further detail on exception handling and other special cases, for example, customers with missing, improbable, or unusual data (Bennett, Skelton & Lunn, 2001; Lockheed Martin, 1996). However, the UML defines a scenario as "a specific sequence of actions that illustrates behaviors" (OMG,