Use Cases in the UML

Brian Dobing
University of Lethbridge, Canada

Jeffrey Parsons
Memorial University of Newfoundland, Canada

INTRODUCTION

The unified modeling language (UML) emerged in the mid-1990s through the combination of previously competing object-oriented systems analysis and design methods, including Booch (1994), Jacobson, Christerson, Jonsson, and Overgaard (1992), Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen (1991) and others. Control over its formal evolution was placed in the hands of the object management group (www.omg.org), which recently oversaw a major revision to UML 2.0 (OMG, 2005). The UML 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 applying the UML in software development.

Use cases play an important role in the unified process, which is frequently described as "use case driven" (e.g., Booch et al., 1999, p. 33). The term "use case" was introduced by Jacobson (1987) to refer to a text document that outlines "a complete course of events in the system, seen from a user's perspective" (Jacobson et al., 1992, p. 157). The concept resembles others being introduced around the same time. Rumbaugh et al. (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.

The official UML 2.0 documentation (OMG, 2005) includes some examples of use case diagrams, which provide an overview that shows which "actors" are involved in each use case. However, the only indication of the "text document" format is that "use cases are typically specified in various idiosyncratic formats such as natural language, tables, trees, etc." (UML, 2005, p. 574). However, virtually every book on the UML offers some format suggestions for use cases (sometimes termed "use case narratives" or "use case descriptions" to clearly distinguish them from diagrams). Together, the use case diagram and narrative are referred to as the "use case model." There are now several books focusing on use cases including Adolph and Bramble (2003), Armour and Miller (2001), Bittner and Spence (2003), Cockburn (2001), Denny (2005), and Övergaard and Palmkvist (2005) along with a few Web sites, notably Cockburn's (http://www.usecases.org). Thus, use cases seem to be well established within the UML despite the lack of any officially endorsed format from the OMG.

BACKGROUND

A use case "describes the system's behavior under various conditions as the system responds to a request from one of the stakeholders" (Cockburn, 2001). A use case should have a clear goal and describe what should happen (but not how it should happen) as users interact with the system. Common examples would include a customer renting a video, purchasing an item, withdrawing funds from a bank account, etc. The use case also identifies the main "actors" involved which, in the previous examples, could include the customer, employees (e.g., rental clerk), a device (bank machine), time (clock), etc. The use case must provide something of value to one or more actors; otherwise there would be no need for it. While the main use case narrative would describe a successful rental, purchase, or withdrawal, alternative outcomes would handle problems such as rejected credit cards, insufficient funds, etc.

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 (with the use case diagrams playing a minor role). Structured analysis emphasized the importance of graphical tools, such as work flow and data flow diagrams. The UML has not abandoned diagrams; thirteen are now included with UML 2. The class, activity, communication (previously collaboration), sequence, state machine (previously statechart), and use case diagrams have always played important roles. But use case narratives are text-based 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 initiation to achievement of the defined goal, from the user's perspective. In particular, a use case has a goal, which comes from the goals of those who will be using the system (Cockburn, 2001). This keeps the focus on the key

requirements and 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 actors can also be external systems or devices. A single use case can involve many actors.

Consistent with an object-oriented approach, use cases can also have generalizations and include and extend relationships. Generalizations allow a child use case to override the behavior of its parent use case in certain situations, but are "not widely used" according to Arlow and Neustadt (2004). An "include" relationship is generally used when the same steps are required by several use cases (e.g., logging into a system), 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). An "extend" relationship exists when a base use case incorporates another use case depending on certain conditions, such as exceptional situations where including the additional detail in the base use case adds too much complexity.

Writing use cases may seem simple enough because they are text-based. However, as discussed in the next section, the content and format of use cases vary somewhat among published books and articles. Those new to use cases would be well advised to read at least a couple of the books devoted to use cases (referenced previously) before incorporating them into a system development project.

ISSUES

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, were intended to be a conceptual tool. The use case should emphasize "what" and not "how" (Jacobson et al., 1994, p. 146). This principle was not strictly followed by much of the early literature, including Jacobson et al. (1992, p. 162) who referred to a display "panel," "receipt button," and "printer" in one of their previous examples. Constantine and Lockwood (2000) distinguish "essential" use cases containing few if any references to technology and user interface implementation, from "concrete" use cases that specify the actual interactions. Others make a similar distinction using the terms "business use cases" and "system use cases." While this provides flexibility, developers need to be careful about what type of content is appropriate at any given time.

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. There also seems to be more acceptance of including exception and error steps, which were less common in earlier books.

Third, the granularity of use cases varies from coarse (few use cases) to fine (many). In principle, use cases should offer "measurable value to an individual actor" (Jacobson et al., 1994, p. 105) and "the collected use cases specify the complete functionality of the system" (White 1994, p. 7). But how to determine the number of use cases this requires is not easily articulated. 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." Övergaard et al. (2005, p.45) suggest the same range (20-50) for "a normal medium-sized system." Armour et al. (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 et al. (2000, p. 125) and Armour et al. (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. Bittner et al. (2003) suggest they are typically 5 to 15 pages, but with 60 to 80% of the content handling exception and error conditions. Jacobson et al. (1999) advocate an iterative development approach in which both the number of use cases and their level of detail increase as the work 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. For example, Kulak et al. (2000) have identified four levels. However, knowing what should be a use case, how much detail is appropriate at each phase, and when to stop are important issues that are difficult to resolve precisely.

To further complicate the issue, some of those who favor fewer or less detailed use cases supplement them with "scenarios." Rumbaugh et al. (2005, p.579) say that "a scenario may be used to illustrate an interaction or the execution of a use case instance." "Add a customer" is a use case. Adding a specified customer with a particular name, address, etc. is a scenario. Others use scenarios to provide further detail on exception handling and other special cases (e.g., customers with missing, improbable, or unusual data) (Bennett, Skelton, & Lunn, 2001) rather than alternative paths in the use case. How many scenarios, alternate paths, and exception paths should be developed, and what their role should be in developing class diagrams, is not clear. A minimalist approach to use cases combined with extensive scenarios and paths may still result in a large and very detailed set of specifications.