An Introduction to the Unified Modeling Language

*A picture is worth a thousand words.*

Most people refer to the Unified Modeling Language as UML. The UML is an international industry standard graphical notation for describing software analysis and designs. When a standardized notation is used, there is little room for misinterpretation and ambiguity. Therefore, standardization provides for efficient communication (a.k.a. “a picture is worth a thousand words”) and leads to fewer errors caused by misunderstanding.

The U in UML stands for unified because the UML is a unification and standardization of earlier modeling notations of Booch, Rumbaugh, Jacobson, Mellor, Shlaer, Coad, and Wirfs-Brock, among others. The UML most closely reflects the combined work of Rumbaugh, Jacobson, and Booch – sometimes called the three amigos. The UML has been accepted as a standard by the Object Management Group\(^1\) (OMG). The OMG is a non-profit organization with about 700 members that sets standards for distributed object-oriented computing.

In this appendix, we bring together for ease of reference five fundamental UML models: use case, class, sequence, state, and activity diagrams. The intent is not for this to be your only UML reference, but to succinctly provide you with the essential 20% of the UML that will provide you with the 80% of the capability you will use often.

1. **Use Case Diagrams**

Use case diagrams are used during *requirements elicitation and analysis* as a graphical means of representing the functional requirements of the system. Use cases are developed during requirements elicitation and are further refined and corrected as they are reviewed (by stakeholders) during analysis. Use cases are also very helpful for *writing acceptance test cases*. The test planner can extract scenarios from the use cases for test cases. Note: The use case diagram is accompanied by a textual use case flow of events. The flow of events is not explained in this document.

A *use case*, a concept invented by Ivar Jacobson (Jacobson, Christerson et al., 1992), is a sequence of transactions performed by a system that yields an outwardly visible, measurable result of value for a particular actor. A use case typically represents a major piece of functionality that is complete from beginning to end (Bruegge and Dutoit, 2000).

In UML, a use case is represented as an ellipse, as shown in Figure 1. In a Monopoly game, some use cases are: Enter Player Info, Buy House, and Draw Card. Give your use case a unique name expressed in a few words (generally no more than five words). These few words must begin with a *present-tense verb phrase in active voice*, stating the action that must take place (notice: **Enter** Player Info, **Buy** House, **Draw** Card, and **Switch** Turn).

---

\(^1\) For more information on the OMG, see [http://www.omg.org](http://www.omg.org)
An actor represents whoever or whatever (person, machine, or other) interacts with the system. The actor is not part of the system itself and represents anyone or anything that must interact with the system to:

- Input information to the system;
- Receive information from the system; or
- Both input information to and receive information from the system.

The total set of actors in a use case model reflects everything that needs to exchange information with the system (Rosenberg and Scott, 1999). In UML, an actor is represented as a stickman, shown below in Figure 2. In a Monopoly game, some actors are the player and a bad player (who has the audacity to want to take two turns in a row!). As you see, actors can be people or they can be other systems. The name of an actor is always a noun. However, the name should not be that of a particular person. Instead, the name should identify the role or set of roles the actor plays relative to one or more use cases.

A use case diagram is a visual representation of the relationships between actors and use cases together that documents the system’s intended behavior. A simple use case diagram is shown in Figure 3.

Arrows and lines are drawn between actors and use cases and between use cases to show their relationships. We discuss these relationships more detail later in this appendix. The default relationship between an actor and a use case is the «communicates» relationship, denoted by a line. For example, in Figure 3, the actor is communicating with the use case.
There are several different kinds of relationships between actors and use cases. Earlier, we said that the default relationship is the «communicates» relationship. The «communicates» relationship indicates that one of these entities initiated invoked a request of the other. An actor communicates with use cases because actors want measurable results. It might not be quite as obvious that use cases can communicate with other use cases. This happens if a case needs information from or to initiate action of another use case. When a line or an arrow is drawn on a diagram and there is no label on the arrow, it is, by default, a «communicates» relationship.

There are two other kinds of relationships between use cases (not between actors and use cases) that you might find useful. These are «include» and «extend». You use the «include» relationship when a chunk of behavior is similar across more than one use case, and you don’t want to keep copying the description of that behavior (Bruegge and Dutoit, 2000). This is similar to breaking out re-used functionality in a program into its own methods that other methods invoke for the functionality. For example, suppose many actions of a system require the user to login to the system before the functionality can be performed. These use cases would include the login use case.

The «include» relationship is not the default relationship. Therefore in a use case diagram, the arrow is labeled with «include» when one use case makes full use of another use case, as shown in Figure 4. The Draw Card and the Buy House both use the View Information functionality.

You use the «extend» relationship when you are describing a variation on normal behavior or behavior that is only executed under certain, stated conditions. The extend relationship is used when the alternative flow is fairly complex and/or multi-stepped,
End of ebook preview
Download the full PDF tutorial from the link below:

Click Here