

UML Sequence Diagrams

by Bruce Powel Douglass

The Unified Modeling Language (UML) is a graphical language for representing system structure and behavior. In a previous article, we looked at UML class diagrams, which are used to represent structure ("UML Class Diagrams," February 2003, p. 39). In this and a subsequent article, we will look at diagrams that can be used to represent behavior.

There are two primary kinds of behavioral views in UML. The first, called sequence diagrams, shows the interactive behavior of collaborations of objects working together (called interactions). The second, called statecharts, shows the behavioral specification of individual objects. This article focuses on the modeling of interactions to capture system behavior via sequence diagrams.

Modeling interactions

Interactions model how objects communicate by sending messages to each other over time. A *stimulus* represents a communication between instances. Formally, *messages* define the communication sent among classes, though most UML users treat stimuli and messages synonymously.

The UML provides two common notations for interactions: *sequence diagrams* and *collaboration diagrams*. Sequence diagrams are far more commonly used than collaboration diagrams and show roughly the same information.

Sequence diagrams can be used in a number of ways during the software development process. During requirements capture, they can help define messages sent between the actors and

the system, their allowable sequences (sometimes called the "protocol of interaction"), and quality of service constraints. During system modeling, sequence diagrams are used to demonstrate how internal structural elements interact to provide higher level behavior, such as to "realize" a use case. And finally, during testing, sequence diagrams can be used to specify expected behavior (given a set of preconditions and an ordered set of stimuli) and validate output.

Figure 1 shows a sequence diagram depicting the interaction of instances of the elements from the class diagram in the article mentioned earlier. Callouts identify the different elements of the sequence diagram. The most important elements are the *object lifelines* and the stimuli (or messages). The object lifelines represent objects playing roles in a collaboration and the stimuli show the messages sent between the lifelines over time. The lifelines represent instances (or roles that instances play) of any classifier in the UML. They are commonly things like objects (instances of classes), actors (objects of interest outside the system), systems, subsystems, components, or even use cases. When a use case is found on a sequence diagram, it represents the part of the system that executes to realize that use case (in other words, the realizing collaboration).

The whole story

Each sequence diagram explicitly shows a legal sequence of messages in the order in which they occur. Well, almost explic-

itly—sequence diagrams specify what is called partial ordering. Because instance lifelines may be running in different threads, the relative order of messages is only fully-ordered in two senses: first, a message-receive event always occurs after that message is sent (no surprise there). Second, all events along a single lifeline are fully ordered in time. However, if two different messages occur on different lifelines, the sequence diagram really doesn't say anything about the relative timing of those particular messages.

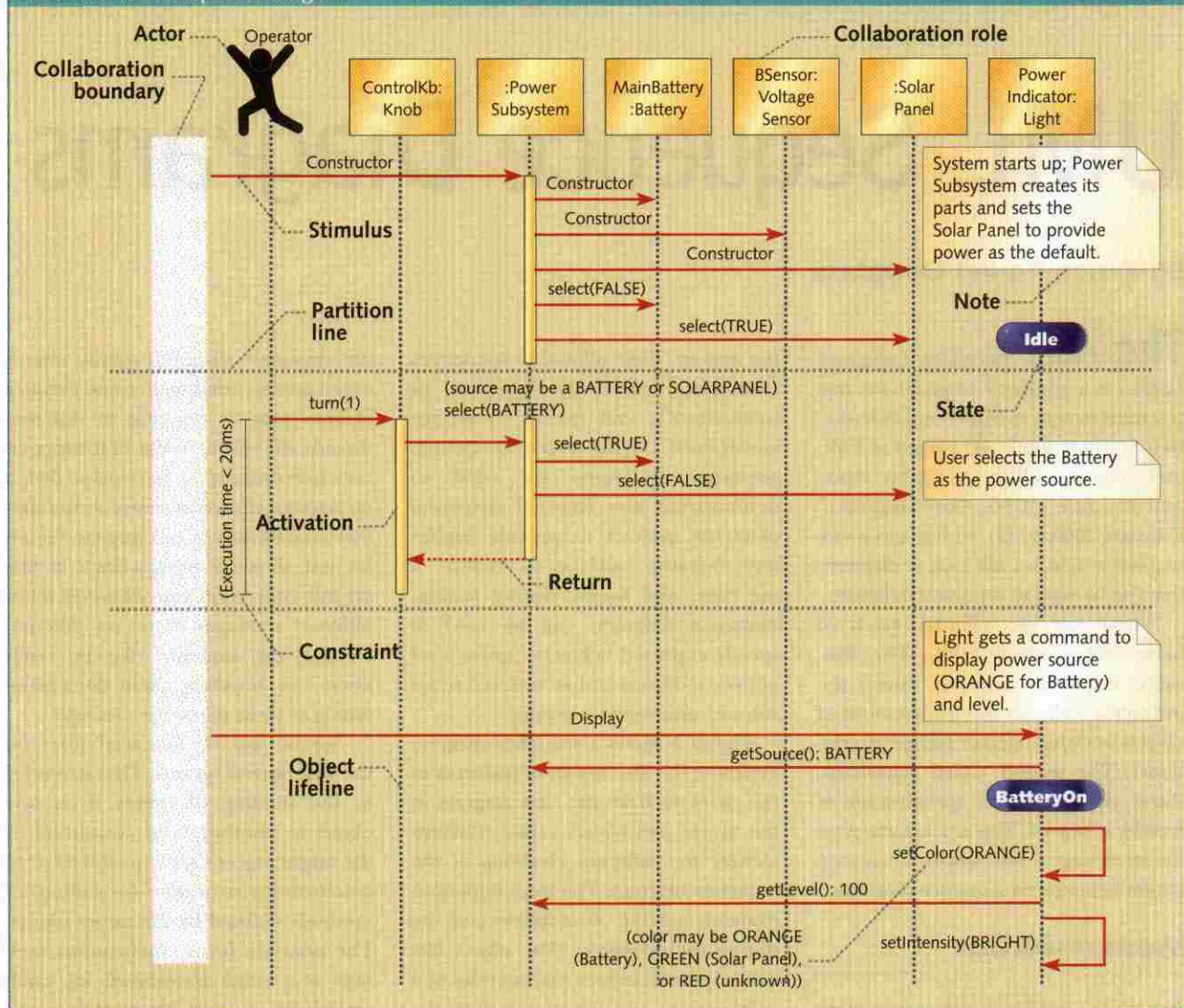
Stimuli are the arrowed lines on the sequence diagrams. They can refer to the sending of events from one object to another (and consumed by the target object's state machine) or to synchronous invocation (or calling) of methods defined by the target object. The notation for a synchronous message is a solid arrowhead; an open arrowhead is used for asynchronous events. The dashed arrows are explicit returns, although they're optional because it's assumed that every call has a matching return.

A sequence diagram shows one possible trace of execution of the collaboration; it's not necessarily the only one. The set and order of messages shown isn't the only possible interaction. The collaboration may well have other interactions.

Other details

Figure 1 shows a few other items that appear on sequence diagrams. The bars on the lifelines are called *activations* and represent the execution of a method

FIGURE 1 A sequence Diagram



provided by that object. Activations are really only useful for synchronous messages, so many people don't use them.

The thick line at the left of the figure is called a *collaboration boundary* and stands for "everything else"—all objects in the universe other than those you've explicitly drawn. This notational shorthand is useful and also provides a place to hook into a testing environment. A test environment plays the role of the collaboration boundary as it stimulates and monitors the objects under test.

The rounded rectangle on the Power Indicator instance lifeline is an indication of that instance's state. The state holds until it receives an event

that causes a change in state, such as the return from the `getSource()` call.

We also see notes and constraints on the figure. A *note* is just uninterpreted free text, usually enclosed in a box with a folded corner. A *constraint* is a user-defined rule of correctness that applies to some set of model elements. The timing constraint in the middle left of the figure specifies an upper boundary for the execution of a portion of the interaction. Other constraints may apply other limitations, such as the set of possible colors on the `setColor()` operation towards the bottom of the diagram.

Because sequence diagrams show a trace of execution, some UML tools can

create them dynamically as a system runs, capturing what the collaboration actually does under a certain set of conditions. This makes automatic validation possible.

Practically speaking, sequence diagrams are a view that binds specification and test together. And now you know how to read them. **esp**

Bruce Powell Douglass is the chief evangelist at I-Logix. He has over 25 years of experience designing safety-critical real-time applications and is the author of several books, including Real-Time UML, Doing Hard Time, and Real-Time Design Patterns. He can be reached at bpd@ilogix.com.

