Building Protocol State Machines in UML 2

When you want to show the sequence of events an object reacts to — and the resulting behavior — you use the UML notation that creates *behavioral state diagrams* (also known as *machines*): Such state diagrams have event/action pairs, entry actions, exit actions, and do activities. Most of your state diagrams use these features; in effect, they are *behavioral state machines.*

Sometimes, however, you just want to show a specified sequence of events that your object responds to — and when it can respond — without having to show its behavior. Such a specified sequence is called an *event protocol.* In UML 2, you can show event protocols by diagramming *protocol state machines.* These differ from behavioral state machines and have special uses.

Normally you should use regular state diagrams to show internal sequences of behavior for all objects of a class. Sometimes, however, you want to show a complex protocol (set of rules governing communication) when using an interface for a class. For example, when you are designing classes that access a database for your application you need to use common operations like open, close and query a database. But these operations must be called in the right order. You cannot query the database before you open it.

One solution to designing a simple database access class is to develop a DatabaseAccessor class with a DBaccess interface as shown in Figure 1. But the DBaccess interface has a complex protocol that governs its use because of the rules governing communication between any other object and the DatabaseAccessor class implementing the DBaccess interface. To use the interface properly, you have to open the database and *then* set up a query. You can put these rules in a state diagram to indicate the protocol that must be followed when using the interface.

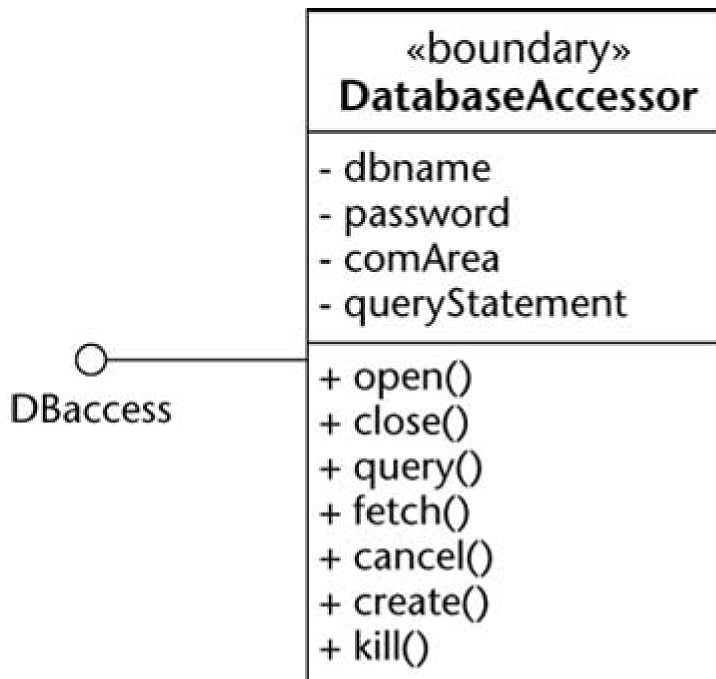

Figure 1: *Class diagram with DBaccess interface.*

Regular state diagrams don't help you with interfaces because interfaces don't describe behavior implementation they just declare what operations the class must perform. It's up to the class to specify the implementation of an interface. On the other hand a protocol state machine enables you to declare what operations can happen and the order they can happen without having to say anything about behavior implementation.

Figure 1 shows the DBaccess interface attached to the DatabaseAccessor class; the DatabaseAccessor class must conform to the operation sequence (that is, the protocol) of the DBaccess interface: The open, close, query, fetch, cancel, create, and kill operations must be implemented in the order specified by the DBaccess interface's protocol (shown in Figure 2).

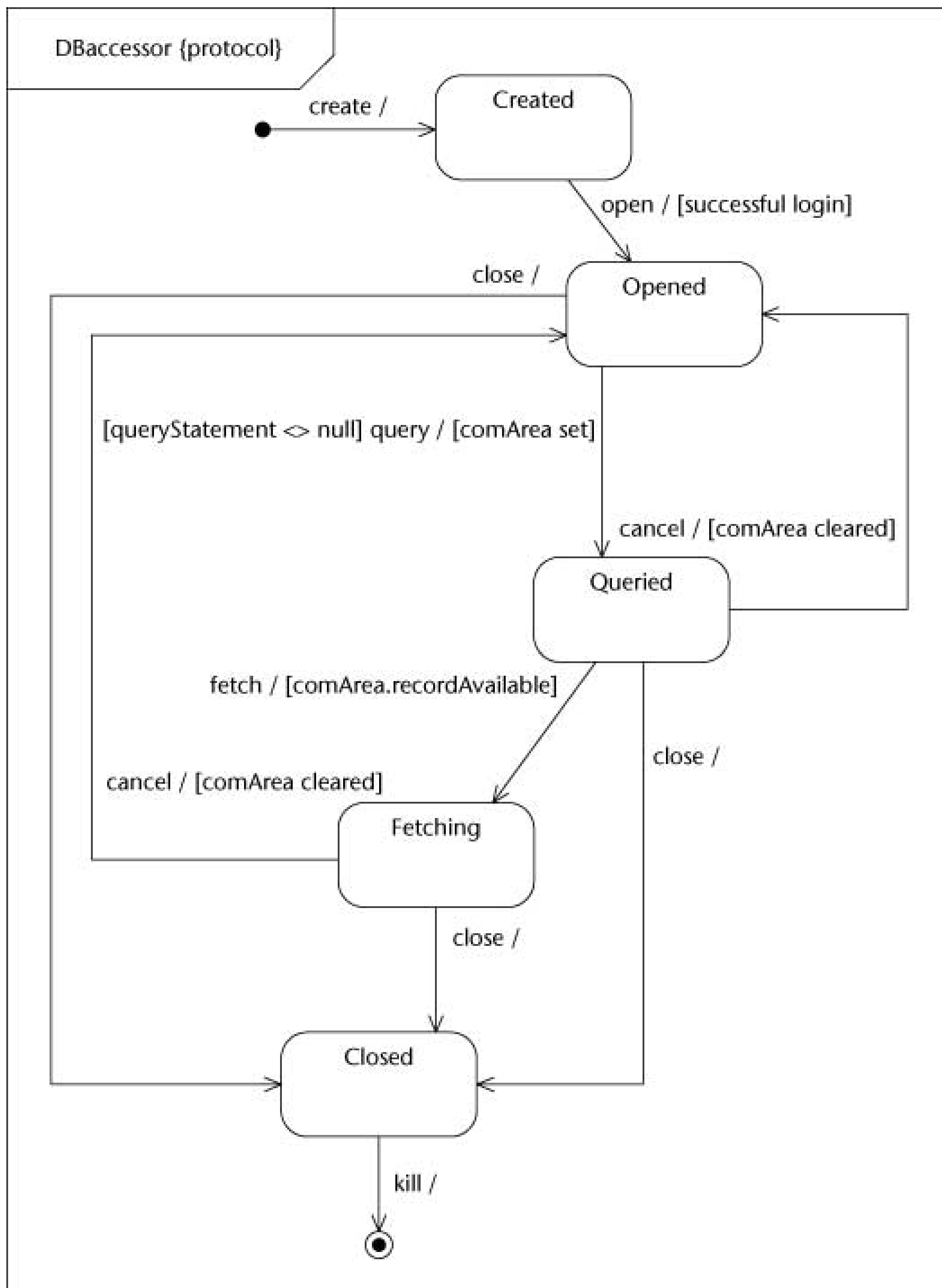

Figure 2: *DBaccessor protocol state machine.*

You draw a protocol state machine in much the same way you draw any other state machine. Remember, however, to follow a few special rules:

- States can have names but can't show entry actions, exit actions, internal actions, or do activities.
- Transitions show operations but not actions or send events (as regular state diagrams can).
- Transitions can have preconditions and postconditions shown in square brackets [], as in the following example:

- [queryStatement <> null] query / [comArea set]
- A *precondition* states what must be true before the object can transition from one state to another. In this example, when an object that conforms to the DBaccessor interface receives the query operation, the queryStatement attribute is checked to see whether it's null. If the object is in the Opened state, and the queryStatement isn't null then the object transitions to the Queried state.
- A *postcondition* states what must be true once the object completes its transition and is now in a new state. In this example, when an object that conforms to the DBaccessor interface makes a successful transition to the Queried state, that means the postcondition must now be true — the comArea is set.
  - You draw your protocol state machine as a group of substates within one large frame.
  - You must name the protocol state machine as such; place the keyword protocol in curly brackets {} next to the name.

The diagram in Figure 2 shows a protocol state machine for the DBaccessor interface. Any class conforming to the DBaccess interface must implement the protocol state machine. You can show the implementation of the protocol state machine as a regular state machine with all the actions and activity behaviors thrown in. That way it's clear to other developers how you will implement the protocol for a specific class in your design.

State diagrams aren't meant to show the flow of *data* from one process step to another. Instead, they're supposed to show where the flow of *control* goes when some behavior happens. Don't let your state diagram mutate into a data-flow diagram.

Copyright © 2009 & Trademark by Wiley Publishing, Inc. All rights reserved.