6.2 Controller States

To understand the design of the Controller Interface we present an analogy. Let us consider the manner in which a radio or a VCR is built. This exercise will give us a good insight into how one can build a multimedia player in software. Broadly speaking to build a hardware player we have to choose a cabinet or housing, and then house appropriate electronics inside that cabinet. The electronics that goes inside the player depends on what we want the radio or the VCR to play. For an AM (amplitude modulated) radio receiver we need certain kind of electronics which is quite different from what we should have for a FM (frequency modulated) radio receiver. Similarly the electronics that handles PAL format in a VCR may not handle the NTSC format. In essence it means that we need to know the kind of media that we will be handling in order to decide the hardware needed to build the player.

6.2.1 Unrealized, Realizing and Realized states

Using the above-mentioned analogy, the first step in the construction of a software player is the instantiation of the Controller object. Normally we do not have classes implementing the Controller interface directly. Instead we only have classes that implement the Player interface which extends the Controller interface. When you instantiate a Player (Controller) it is like a cabinet or housing. At this stage the Controller has no knowledge of the media it is supposed to handle. Therefore the Controller cannot identify the resources it has to acquire in order to handle the media. This state of the Controller is called as the “Unrealized” state because the Controller has not yet realized the media it has to handle.

The next step is to make the Controller realize the media it has to handle and identify the resources required to handle the media. To make a Controller realize we use the method,

public void realize().

The process of realization may be time consuming. During realization the Controller not only identifies the resources it needs, but may also acquire all those resources excepting for the exclusive-use and scarce resources. During the realization process we say that the Controller is in the “Realizing” state. Once the realization process is over the Controller moves from the Realizing state to the Realized state. The main reason for not acquiring the scarce and exclusive-use resources during realization is that we may not start the Controller immediately after realization. It is not wise to tie up the exclusive-use resources while they are not in use. Doing so will block all those applications that are waiting to acquire the exclusive-use resources.

6.2.2 Prefetching and Prefetched state

Suppose we want a Realized Controller to start handling the media either for presentation or for processing. In order to get started the Controller has to acquire the exclusive–use resources needed by it. Further it should also fetch the media data. Acquiring the exclusive–use resources and fetching the media data are generally time-consuming operations. This is especially prominent when the source of the media is remote and therefore the media has to travel over the network.

To solve this problem the Controller has a method called Prefetch() which can be invoked on a Realized Controller. When the method Prefetch() is invoked the Controller enters into the

Prefetching state. In the Prefetching state the Controller acquires the exclusive-use resources as well as prefetches the media data and fills its buffer with the media. Once the prefetching process is over the Controller moves from the Prefetching state to the Prefetched state. A Prefetched Player is ready to start and can be started with the minimal possible latency.

6.2.3 Starting and stopping a Controller

To start a Prefetched Controller you can use the syncstart() method inherited from the Clock interface. Suppose a Controller is in the started state then the Controller will move to the stopped state under one of the following conditions.

(I) the end of the media is reached or
(II) the stopTime (set by setStopTime() method) has been reached or
(III) the stop() method is invoked.

6.2.4 Deallocating and closing a Controller.

The method deallocate() aborts the current operation and releases any exclusive use resource that the Controller might have acquired. The state of the Controller returns to the realized state if the method deallocate() is called after realization. The state returns to the unrealized state if the method deallocate() is called during realization. The method close() releases all the resources and ceases all the activities of the Controller. Both the methods deallocate() and close() releases the Controller's exclusive use resources. However the distinction between these two methods is as follows. After a close operation the Controller can no longer be used. On the other hand after the deallocate operation the Controller will be either in the Unrealized state or in the Realized state and can still be used.

At any given time the Controller is in a particular state and is in the process of moving to another state. The current state and the target state of the Controller can be found out respectively by using the methods,

public int getState(),

and

public int getTargetState().

6.2.5 Startup latency

The startup latency is the maximum amount of time the Controller takes to present the first frame of the media from the instant the method syncStart() is invoked. The startup latency can be queried using the Player's method

public Time getStartLatency( ).

The knowledge of startup latency is helpful in certain situations. Suppose you want the presentation of the media to start at a specified time instant. You should then invoke the method syncStart() in advance to the specified instant by a time duration equal to the startup latency.